SEMANTIC ADVANCED MATCHMAKER
(SAM)

by

Erdem Savaş İlhan

B.S., Computer Engineering, Boğaziçi University, 2004

Submitted to the Institute of Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2007

ACKNOWLEDGEMENTS

I would like to thank and state my sincere gratitude to many people, who have contributed and supported throughout this research. First of all, I would like to thank to my thesis advisor Assistant Prof. Ayşe Başar Bener for her good advice, guidance and for giving me the proper motivation during this research. I would also like to thank Assistant Prof. Pınar Yolum Birbil for her valuable comments and many thanks to all colleagues in Web Services Research Group.

I should also state my gratitude to my colleagues and managers at work for their understanding throughout this study.

Finally, I would like to thank to my family and fiancée Gökçe for their endless moral support and love.

ABSTRACT

SEMANTIC ADVANCED MATCHMAKER (SAM)

As the number of available Web services increase finding appropriate Web services to fulfill a given request becomes an important task. Most of the current solutions and approaches in Web service discovery are limited in the sense that they are strictly defined, and they do not use the full power of semantic and ontological representation. Service matchmaking, which deals with similarity between service definitions, is highly important for an effective discovery. Studies have shown that use of semantic Web technologies improves the efficiency and accuracy of matchmaking process.

In this research we focus on one of the most challenging tasks in service discovery and composition: Service matchmaking. We make use of current semantic Web technologies like OWL and OWL-S to describe services and define ontologies. We introduce an efficient matchmaking algorithm based on bipartite graphs. We have seen that bipartite matchmaking has advantages over other approaches in the literature for parameter pairing problem, which deals with finding the semantically matching parameters in a service pair. Our proposed algorithm ranks the services in a candidate set according to their semantic similarity to a certain request. Our matchmaker performs the semantic similarity assignment implementing the following approaches: Subsumption-based similarity, property-level similarity, similarity distance annotations and WordNet-based similarity. Our results show that the proposed matchmaker enhances the captured semantic similarity, providing a fine-grained approach in semantic matchmaking.

ÖZET

GELİŞMİŞ ANLAMSAL WEB SERVİS EŞLEYİCİ (SAM)

Elektronik ortamda bulunan Web servislerinin sayısı arttıkça belirli bir isteği karşılayacak uygun servisleri bulmak da önem kazanmaktadır. Bu probleme ilişkin günümüzdeki çözümler ve yaklaşımlar kısıtlı tanımlamalar kullandıkları için ve anlamsal tanımlamalardan tam anlamıyla faydalanmadıklarından yetersiz kalmaktadırlar. Servis tanımlamaları arasındaki benzerlikleri bulmada kullanılan servis eşleme yöntemi etkin bir Web servis bulma işlemi için büyük önem taşımaktadır. Günümüze kadar olan çalışmalar göstermiştir ki anlamsal Web teknolojilerinin kullanımı servis eşleme yönteminin etkinliğini ve başarısını artırmaktadır.

Bu araştırmada Web servisi bulma ve kompozisyonu konuları için çok önemli olan bir konuyu inceliyoruz: Web servis eşlemesi. Bu çalışmada OWL ve OWL-S gibi servisleri ve ontolojileri ifade etmede kullanılan güncel anlamsal Web teknolojilerini kullanıyoruz. Ayrıca, iki kümeli grafiklere dayanan etkin bir Web servis eşleme algoritmasını tanıtıyoruz. Güncel çalışmaların çoğunda da görüldüğü üzere Web servislerinde parametrelerin anlamsal olarak eşlenmesi problemi olan parametre eşlemesi için iki kümeli grafiklere dayalı çözümler çok daha etkindir. Önerdiğimiz eşleme algoritması belirli bir isteğe denk gelen aday Web servis kümesi içinden isteğe en benzer olan servisleri sıralayarak çıktı olarak sunmaktadır. Ortaya koyduğumuz Web servis eşleyicisi anlamsal benzerlik eşlemesi işlemi için şu teknikleri kullanmaktadır: Kapsama tabanlı benzerlik, özellik tabanlı benzerlik, benzerlik uzaklığı notları ve WordNet tabanlı benzerlik. Elde ettiğimiz sonuçlar gösteriyor ki ortaya koyduğumuz servis eşleyici anlamsal ilişkilerin ortaya çıkarılmasında daha etkin olmakta ve detaylı bir eşleme yapılmasına imkan vermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
CORBA	Common Object Request Broker Architecture
DAML	DARPA Agent Markup Language
DCOM	Distributed Component Object Model
DTD	Document Type Definition
HTTP	Hypertext Transport Protocol
OASIS	Organization for the Advancement of Structured Information Standards
OWL	Web Ontology Language
OWL-S	OWL based Web service Ontology
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RPC	Remote Procedure Call
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SWRL	Semantic Web Rule Language
UDDI	Universal Description, Discovery, Integration
URI	Universal Resource Identifier
WSDL	Web Services Description Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1. INTRODUCTION

1.1. Motivation

In recent years, Web services became the dominant technology in providing the interoperability among different systems throughout the Web. If Web service is used in limited business domain or with strict rules with known business partners everything will be fine. The problem of finding the right and most suitable Web services for user needs emerges when open e-commerce systems are widely used and user requirements dynamically change over time.

Although there are currently proposed technologies for discovery of Web services, such as UDDI [5], they do not satisfy the full discovery requirements. This discovery process is based on syntactical matching and keyword search that does not allow the automatic processing of Web services. To solve the problem of automatic discovery and processing of Web services, the Semantic Web [6] vision is proposed. Semantic Web is an effort by the W3C consortium [7], and one of its main purposes is to facilitate the discovery of Web resources.

There are different efforts and frameworks for semantic annotation and discovery of Web services [10, 11, 12]. For Web service discovery the researchers also propose some techniques and algorithms. However, they mostly classify the discovered Web services as set-based. They do not focus on rating the Web services using semantic distance information [13].

The evolution of Web services, from conventional services to semantic services, caused service descriptions contain extra information about functional or non-functional properties of Web services. The semantic information included in the service descriptions enables the development of advanced matchmaking schemes that are capable of assigning degrees of match to the discovered services. Semantic discovery of Web services means semantic reasoning over a knowledge base, where a goal describes the required Web service capability as input. Semantic discovery adds accuracy to the search results in comparison to traditional Web service discovery techniques, which are based on syntactical searches over keywords contained in the Web service descriptions [3].

Improvement in matching process could be gained by the use of ontological information in a useful form. With the use of this information, it would be possible to rate the services found in discovery process. As in real life, users/ agents should be able to define how they see the relation of ontological concepts from their own perspective. Similarity measures have been widely used in information systems [14, 15, 16], software engineering [68, 69] and AI [17, 18, 19]. So integration of knowledge from these techniques can improve the matching process.

By using semantic distance definition information, we aim to get a rated and ordered set of Web services as the general result of the discovery process. We believe that this would be better than set-based classification of discovered services. In this research, we propose a new scheme of matchmaking that aims to improve retrieval effectiveness of semantic matchmaking process. Our main argument is that conventional evaluation schemes do not fully capture the added value of service semantics and they do not consider the assigned degrees of match, which are supported by the majority of discovery engines. The existing approach to service matchmaking contains subsumption values regarding the concept that the service supports. In our proposed approach, we add semantic relatedness values onto existing subsumption-based procedures. Our matchmaker performs the semantic similarity assignment implementing the following value-added approaches: Subsumption-based similarity, property-level similarity, similarity distance annotations and WordNet-based similarity.

1.2. Outline

In Section 2, we describe enabling technologies for Web services: XML, SOAP, WSDL and UDDI. We discuss on the emerging approach SOA (Service Oriented Architecture) and point out the weak points in current standards. We will introduce semantic Web technologies as an approach to extend capabilities of current SOA technologies. RDF, OWL, OWL-S and some existing semantic Web technologies, as well as some semantic web frameworks are presented in this section. We also give a brief introduction of bipartite graph matching algorithms and description logics which form the basis in our proposed model.

In Section 3, we refer to related work in this field and point out the contributions they make, emphasizing how they can be extended in a way to support better results. In section 4 we state the problem that we focus on in this research. We list the problems in semantic service matchmaking that we are seeking answers for.

Section 5 presents our approach to service matchmaking. We give details of our proposed approach and introduce the techniques and methods we apply in formalism. Section 6 presents the evaluation results of the test runs which we performed on a prototype system that we implemented. We present numerical results on test runs and discuss these results, clearly demonstrating the advantages of our proposed approach with a comparison to a conventional service matchmaker. Finally, Section 7 concludes our research with the future directions for extending the capabilities of our service matchmaker.

2. Web Services and Semantic Web

2.1. Service Oriented Architecture (SOA)

According to Hashimi [45], in SOA, software applications are built on basic components called services, defined in terms of what it does, typically carrying out a business-level operation.

A service in SOA is an exposed piece of functionality with three properties [45]:

1. The interface contract to the service is platform independent.
2. The service can be dynamically located and invoked.
3. The service is self-contained. That is, the service maintains its own state.

There are basically three functions that must be supported in a service-oriented architecture [45]:

1. Describe and Publish service
2. Discover a service
3. Consume/interact with a service

OASIS (Organization for the Advancement of Structured Information Standards) defines SOA as the following [70]:

"A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations".

SOA is not only a set of technologies but it actually represents a shift in the way software operates. SOA represents systems that are self-contained, loosely-coupled and dynamically invoked. Adopting SOA is essential to deliver the business agility and IT flexibility promised by Web Services [47].

2.2. Web Services Standards

Web services enable seamless interoperation among applications and organizations in a distributed environment, while being platform and language independent.

Distributed programming approaches like CORBA and DCOM have been used to implement network enabled applications. However, these technologies depend on binary standards and custom interface definition languages. On the other hand, Web services depend on textual standards and make use of XML in interface definition, which provides platform and language independency.

In a typical scenario, as represented in Figure 2.1, a service requestor communicates with a service registry (UDDI) to discover a Web service satisfying his needs [71]. The service provider publishes his service in the registry to be discovered by clients. Following the discovery phase, the requestor and the provider communicates directly using SOAP, which is a communication protocol based on XML.

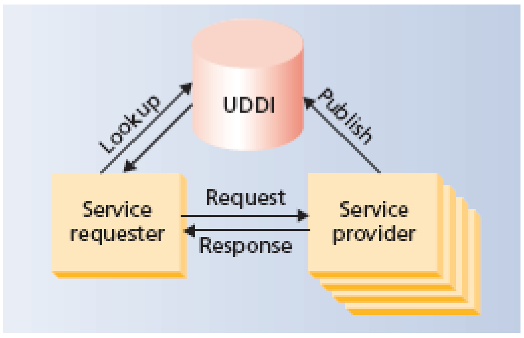


Figure 2.1. Web services architecture [71]

Web services enable both document oriented and RPC (Remote Procedure Call) style communication. In RPC style, Web services can be used to invoke operations on the provider site, whereas document oriented communication provides arbitrary information to be sent to the receiving party. Web services also can operate both synchronously and asynchronously.

Several technologies have emerged in years that enable Web services architecture. As our research builds on top of Web service standards and semantic web standards, in the following sections we describe these technologies in more detail.

2.2.1. XML

Extensible Markup Language (XML) provides the description, storage and transmission format for data exchanged via Web services. Originally designed to meet the challenges of large-scale electronic publishing, XML has also been playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere [51].

XML data is expressed in textual format as opposed to binary encoding which makes it platform and language independent. These properties of XML make it a good approach for communicating in heterogeneous distributed environments. The structure and relationships within the data are expressed with a DTD (Document Type Definition) or XML schema. XML schema language is used to describe the structure and content of an XML document. This ability lets computer programs validate the XML data to ensure that it is structured appropriately [52].

XML technology is one of the core technologies that our research builds on. Ontology languages like RDF and OWL are represented with XML and benefits from the XML standards like namespaces, XML Schema etc.

2.2.2. WSDL

WSDL is an XML grammar describing Web services as a collection of network endpoints, or ports [48]. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations [48]. WSDL is often used in combination with SOAP and XML Schema to provide Web services over the Internet. A client program connecting to a Web service can read the WSDL to determine what functions are available on the server. Any special data types used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the functions listed in the WSDL.

2.2.3. SOAP

SOAP provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment [49]. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics [50].

The SOAP specification defines the envelope structure, encoding rules, and conventions for representing Web service invocations and responses. The preferred transmission method is SOAP over HTTP. The SOAP envelope is the outermost element and indicates the start and end of the message. SOAP envelope basically provides the necessary encapsulation.

The SOAP header element provides the flexibility to include additional mechanisms like security, quality of service and other meta-data related information. Header element comes just after the envelope declaration. SOAP body contains the actual transmitted XML data. It comes after the header element declarations. The format of the SOAP body is defined in the SOAP schema. Fault handling is provided with the SOAP fault element in the body. The receiver of the message resolves the error processing the fault tags in the body [50].

2.2.4. UDDI

The Universal Description, Discovery, and Integration (UDDI) protocol represents the service registry component in the SOA pattern. It defines a standard method of publishing and discovering services in a service oriented architecture. Its development is led by the OASIS consortium [59].

UDDI describes a registry of Web services and programmatic interfaces for publishing, retrieving, and managing information about services. The UDDI specification defines services that support the description and discovery of businesses, organizations, and other Web services providers, the Web services they make available, and the technical interfaces which may be used to access and manage those services. UDDI is base upon several other established industry standards, including HTTP, XML, XML Schema (XSD), SOAP, and WSDL [59].

2.3. Semantic Web

As Tim Barners-Lee stated in 1999, "*The first step is putting data on the Web in a form that machines can naturally understand, or converting it to that form. This creates what I call a Semantic Web-a Web of data that can be processed directly or indirectly by machines"*[35].

Current Web structure puts emphasize on the applications that process data, not the data itself. So, how data is organized and processed varies between applications. This prevents a collaborative and autonomous Web to evolve. Semantic Web is a vision to change current Web structure and make it data driven. Semantic Web technologies could be used in many ways to transform the functionality of the Web [36]: Rich metadata for media and content to improve search and management; rich descriptions of Web services to improve discovery and composition; common access wrappers for information systems to simplify integration of disparate systems; common lingua franca for exchange of semantically rich information between active software agents.

Semantic Web will basically allow autonomous agents to process heterogeneous data on the Web on behalf of a user. In order to achieve this data should be organized in a standard and machine interpretable way. This is achieved through the adoption of common conceptualizations referred to as *ontologies* [37]. Several ontologies are being developed today, both in academia and industry. These constitute the initial steps towards realizing semantic Web vision. Ontology Web Language (OWL) is the most promising standard for developing ontologies. We will describe OWL in detail in the following sections.

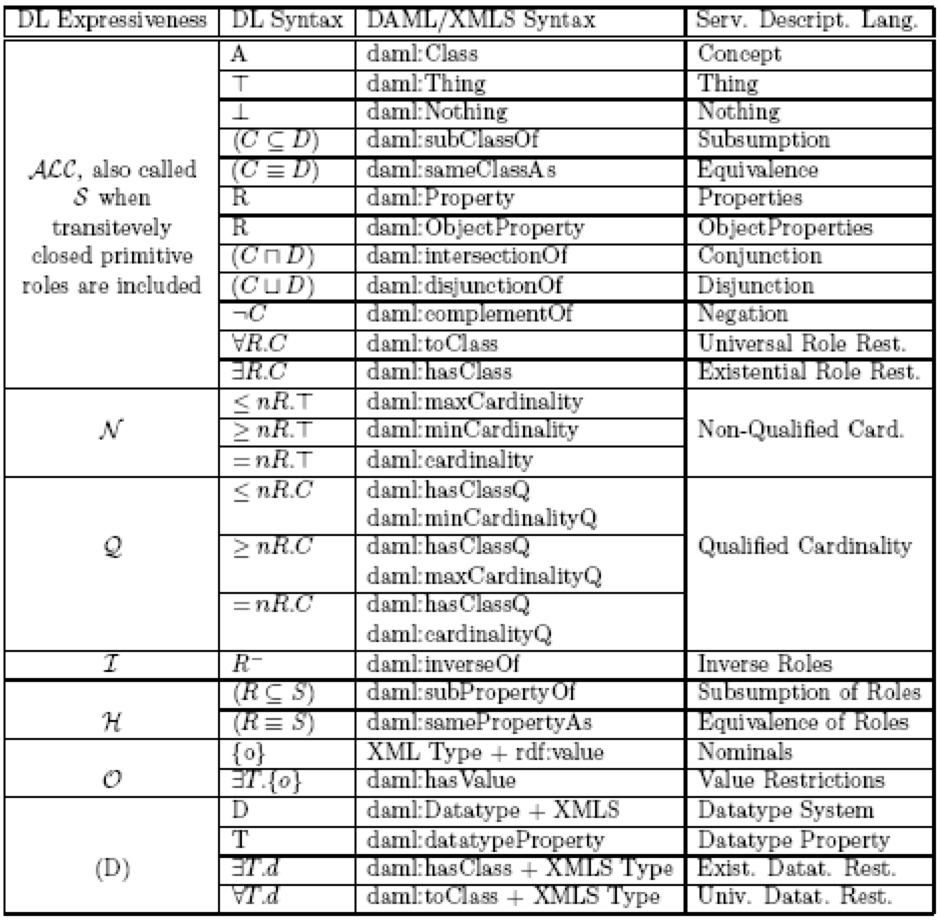
2.3.1. Description Logics

Description logics (DL) are a family of knowledge representation languages which can be used to represent knowledge on a domain and make use of concepts as the building blocks.

Description logics form the basis in the design of ontologies in Semantic Web. OWL endorsed by W3C is a language based on DL. In DL, a concept represents set of objects, whereas roles link objects in these sets in a binary relation form. The basic reasoning problems for concepts in a DL are satisfiability, which accounts for the internal coherency of the description of a concept (no contradictory properties are present), and subsumption, which accounts for the more general/more specific relation among concepts [67].

In DLs, a distinction is provided between TBox (Terminological Box) and ABox (Assertional Box). TBox represents the information on concept hierarchies and relations between them, whereas ABox represents the individuals and their relations to concepts. The separation is useful in both modeling an ontology and in processing to achieve inferencing.

In [34] the authors provide a mapping between description logics and DAML (predecessor of OWL) expressivity in a useful form. The subset of OWL, OWL-DL, is defined as SHOIN$^+$(D) according to the below expressivity table. Figure 2.2 lists this mapping.



DL Expressiveness	DL Syntax	DAML/XMLS Syntax	Serv. Descript. Lang.
$\mathcal{ALC}$, also called $\mathcal{S}$ when transitevely closed primitive roles are included	A	daml:Class	Concept
	$\top$	daml:Thing	Thing
	$\bot$	daml:Nothing	Nothing
	$(C \subseteq D)$	daml:subClassOf	Subsumption
	$(C \equiv D)$	daml:sameClassAs	Equivalence
	R	daml:Property	Properties
	R	daml:ObjectProperty	ObjectProperties
	$(C \sqcap D)$	daml:intersectionOf	Conjunction
	$(C \sqcup D)$	daml:disjunctionOf	Disjunction
	$\neg C$	daml:complementOf	Negation
	$\forall R.C$	daml:toClass	Universal Role Rest.
	$\exists R.C$	daml:hasClass	Existential Role Rest.
$\mathcal{N}$	$\leq nR.\top$	daml:maxCardinality	Non-Qualified Card.
	$\geq nR.\top$	daml:minCardinality	
	$= nR.\top$	daml:cardinality	
$\mathcal{Q}$	$\leq nR.C$	daml:hasClassQ daml:minCardinalityQ	Qualified Cardinality
	$\geq nR.C$	daml:hasClassQ daml:maxCardinalityQ	
	$= nR.C$	daml:hasClassQ daml:cardinalityQ	
$\mathcal{I}$	R^-	daml:inverseOf	Inverse Roles
$\mathcal{H}$	$(R \subseteq S)$	daml:subPropertyOf	Subsumption of Roles
	$(R \equiv S)$	daml:samePropertyAs	Equivalence of Roles
$\mathcal{O}$	$\{o\}$	XML Type + rdf:value	Nominals
	$\exists T.\{o\}$	daml:hasValue	Value Restrictions
(D)	D	daml:Datatype + XMLS	Datatype System
	T	daml:datatypeProperty	Datatype Property
	$\exists T.d$	daml:hasClass + XMLS Type	Exist. Datat. Rest.
	$\forall T.d$	daml:toClass + XMLS Type	Univ. Datat. Rest.

Figure 2.2 Expressivity mapping between DL and DAML [34]

2.3.2. RDF

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources [33].

RDF provides a framework that enables the information to be processed by applications without loss of meaning. It was first defined by W3C in 1997. In 1999 RDF became a W3C recommendation. RDF consists of entities, represented by unique identifiers and binary relationships, or statements, between those entities [38]. In terms of triple representation, the source of the relationship is the *subject*, the relationship itself is the *predicate* and the destination of the relationship is called the *object*. RDF data model also distinguishes resources, which are identified by URIs and literals which are simple strings. The subject and the predicate are always resources, whereas object can be either resource or literal.

RDF assigns each resource a URI [37]. Figure 2.3 shows a representation of a person, Eric Miller, in RDF. Both the nodes in the graph and the arcs between them are associated with URIs. Figure 3 represents:

1. Individuals—such as Eric Miller, identified by http://www.w3.org/ People/EM/contact#me.
2. Kinds of things—such as Person, identified by http://www.w3.org/ 2000/10/swap/pim/contact#Person.
3. Properties of those things—such as mailbox, identified by http:// www.w3.org/2000/10/swap/pim/contact#mailbox.
4. Values of those properties—such as mailto:em@w3.org as the value of the mailbox property (RDF also uses character strings such as "Eric Miller" and values from other data types such as integers and dates as property values).

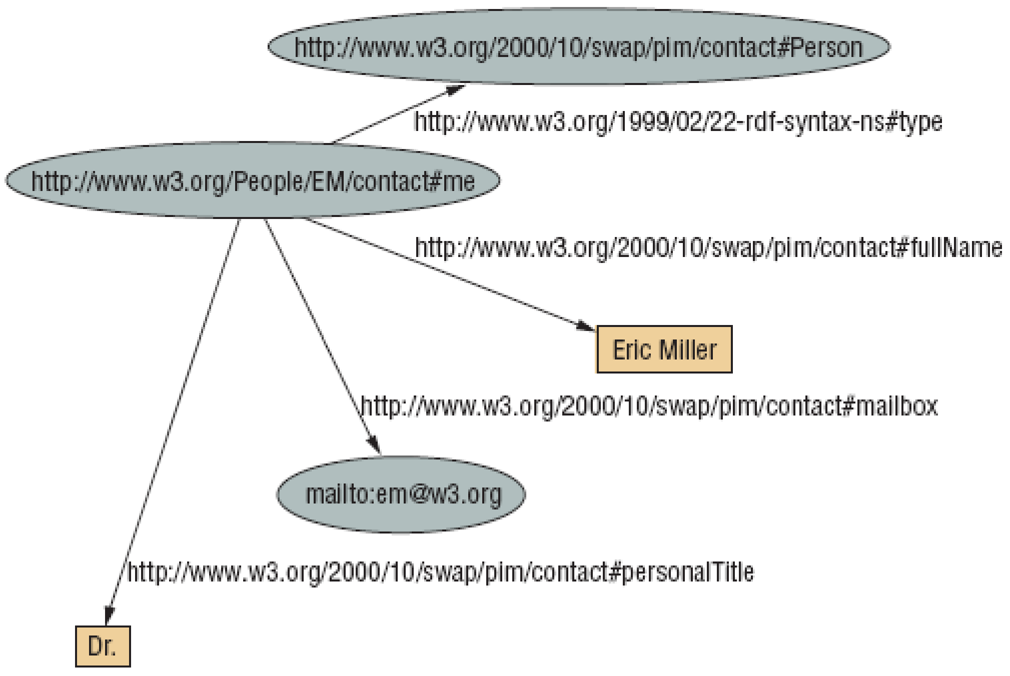

http://www.w3.org/2000/10/swap/pim/contact#Person
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/People/EM/contact#me
http://www.w3.org/2000/10/swap/pim/contact#fullName
Eric Miller
http://www.w3.org/2000/10/swap/pim/contact#mailbox
mailto:em@w3.org
http://www.w3.org/2000/10/swap/pim/contact#personalTitle
Dr.

Figure 2.3. An RDF graph representing Eric Miller [33]

URIs play an important role in RDF as they identify resources. URIs provide a global naming convention that can map to a resource on the Web. RDF Schema provides a contract between the resource provider and the consumer, guaranteeing the applications to work with a RDF instance complying with the schema. It provides means to define property domains and ranges, and class and subclass hierarchies [38].

2.3.3. OWL

Web Ontology Language (OWL) facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics [39]. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full. The choice between OWL Lite and OWL DL depends on the extent to which users require the more expressive restriction constructs provided by OWL DL. Reasoners for OWL Lite will have desirable computational properties. Reasoners for OWL DL, while dealing with a decidable sublanguage, will be subject to higher worst-case complexity. The choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modelling facilities of RDF Schema (i.e. defining classes of classes). When using OWL Full as compared to OWL DL, reasoning support is less predictable [39].

OWL makes an open world assumption, which means that the resources are not restricted to a single scope. A certain concept can be defined in an ontology, whereas another ontology can redefine it with some differences. However, the system should detect any conflicts between assertions in different ontologies.

OWL ontology may include descriptions of classes, properties and their instances. Given such ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics [40]. A typical OWL ontology begins with a namespace declaration similar to the one in Table 2.1.

Table 2.1. OWL namespace declaration

```
<rdf:RDF
   xmlns     ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
   xmlns:vin ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
   xml:base  ="http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#"
   xmlns:food="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food#"
   xmlns:owl ="http://www.w3.org/2002/07/owl#"
   xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
```

OWL depends on constructs defined by RDF, RDFS, and XML Schema data types [40]. Assertions about the ontology are represented in *owl:Ontology* tag. Table 2.2 shows an example OWL ontology description.

Table 2.2. OWL ontology description

```
<owl:Ontology rdf:about="">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <owl:priorVersion rdf:resource="http://www.w3.org/TR/2003/PR-owl-guide-20031215/wine"/>
  <owl:imports rdf:resource="http://www.w3.org/TR/2004/REC-owl-guide-20040210/food"/>
  <rdfs:label>Wine Ontology</rdfs:label>
  ...
```

Concepts are defined with *owl:Class* tags in OWL. Every individual in the OWL world is a member of the class *owl:Thing*. Table 2.3 shows a simple class description.

Table 2.3. OWL class description

```
<owl:Class rdf:ID="Winery"/>
```

Members of the concepts are referred as individuals and represented as having a type of a certain class. Table 2.4 shows an individual description. The *CentralCoastRegion* is declared to be of type *Region.*

Table 2.4. OWL individual description

```
<owl:Thing rdf:ID="CentralCoastRegion" />

<owl:Thing rdf:about="#CentralCoastRegion">
  <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

Properties describe additional information on concepts. Following type of properties are defined in OWL: *ObjectProperty*, *DatatypeProperty*, Object properties have ranges as concepts whereas data type properties have ranges as literals. The domain of the property represents the concept that the property is associated to and the range represents the target object/literal that the property refers to. Table 2.5 shows a property named *madeFromGrape,* which is a property of *Wine* class and has range of type *WineGrape* [40].

Table 2.5. OWL property description

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

2.3.4. OWL-S

OWL-S is an OWL-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-intepretable form. OWL-S markup of Web services will facilitate the automation of Web service tasks, including automated Web service discovery, execution, composition and interoperation [41]. OWL-S is indeed a set of top-level ontologies written in OWL specifically for the description of Web services.

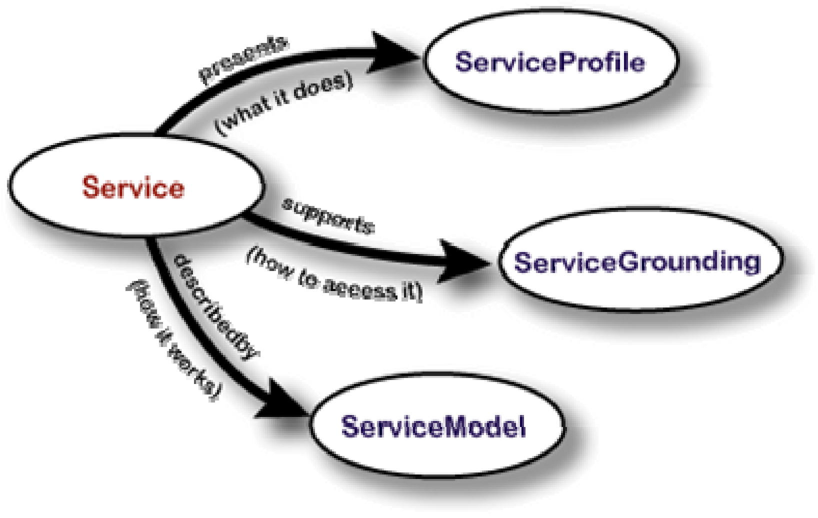


Figure 2.4. OWL-S ontologies [42]

OWL-S is composed of the following top-level ontologies as shown in Figure 2.4: Service Profile, Service Model and Service Grounding [42]. Service profile describes "what the service does", providing the capabilities of the service in an interface format. Service input, output parameters with the preconditions and effects are described in the service profile in a brief form. In service discovery profiles are applied in two ways. On one hand, they are used by service providers to publish Web services. These profiles are called *advertisements*. On the other hand, profiles are used by a service requester to describe the Web service to be searched for. During discovery this *request* is compared with published advertisements to find suitable services [43]. Additionally, non-functional parameters like the service name and description or quality of service information can be included in the service profile section.

Service model details the process flow of the service. It uses the control constructs such as Sequence, Split, Split + Join, Choice, Any-Order, Condition, If-Then-Else, Iterate, Repeat-While, and Repeat-Until as in a conventional process modeling language to define composite services. Service model details the information on inputs, outputs, preconditions and effects of the service with the types in ontologies they refer to [42]. Service grounding specifies how an agent can access the service. It details information on communication protocol, message formats and other information such as port numbers etc [42]. The upper ontology for services specifies only two cardinality constraints: a service can be described by at most one service model, and grounding must be associated with exactly one service [43].

OWL-S defines three classes of processes. Atomic processes (AtomicProcess) are directly executable and contain no further sub-processes. From the view of the caller atomic processes are executed in a single step which corresponds to the invocation of a Web service method. Simple processes (SimpleProcess) are not executable. They are used to specify abstract views of concrete processes by hiding certain IOPEs. Composite processes (CompositeProcess) are specified through composition of atomic, simple and composite processes recursively by referring to control constructs (ControlConstruct) using the property composedOf. Control constructs define specific execution orderings on the contained processes [43]

2.4. Semantic Web Frameworks

2.4.1. Jena

Jena is a Java framework for building semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine [29]. It is an open-source framework supported by HP Semantic Labs Programme.

As we reason on OWL ontologies, in our research we are mostly interested in the ontology API and inference support that Jena provides. Jena is at the core an RDF framework and supports ontology formalisms built on top of RDF. However, Jena Ontology API is independent of the ontology language being processed and provides a consistent interface. The class names, *OntClass* and *ObjectProperty* support this transparency. Profiles are used to differentiate underlying ontology languages. Thus, in a daml ontology property name for *ObjectProperty* is *daml: ObjectProperty*, whereas it is *owl ObjectProperty* in owl [29].

As seen in Figure 2.5, Jena *Model* interface operates on the RDF data collection which is represented as a graph. The bottom layer represents the asserted statements, whereas the middle layer holds the inferred statements. The inference is performed by the reasoner and operates on the base graph by use of semantic rules of the underlying language.

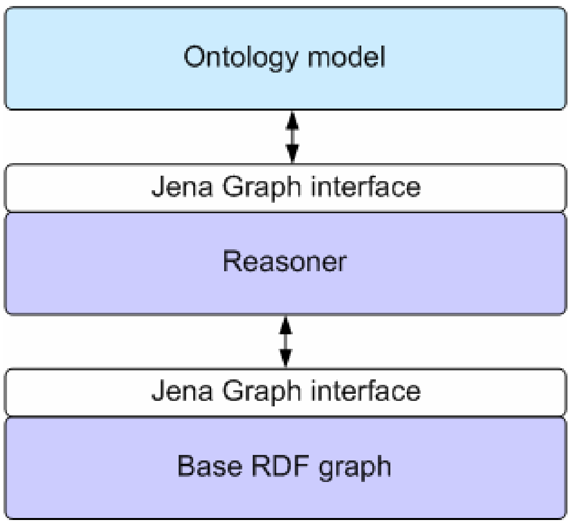


Figure 2.5. Jena model interface [29]

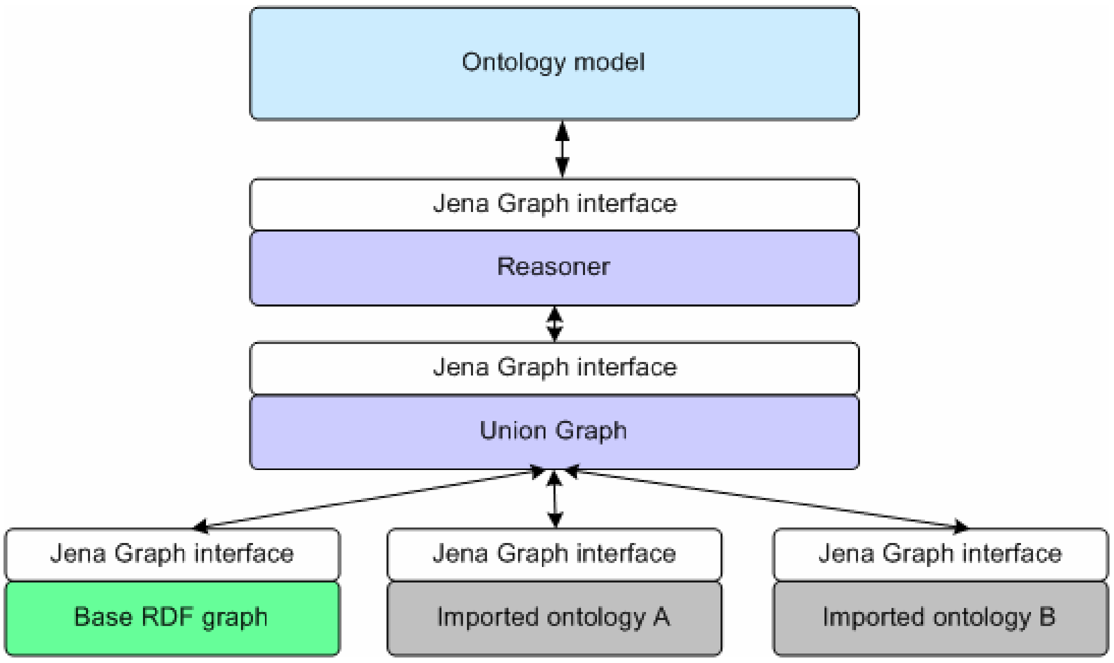


Figure 2.6. Jena ontology internal structure with imports [29]

As Figure 2.6 presents, Jena handles the imported ontologies by unifying them in the underlying graph model. This helps to reason on multiple ontologies without loss of information.

Inference in Jena follows the structure in Figure 2.7. The reasoner to be used can be plugged in and out using the *ReasonerFactory*. The reasoner works on the base RDF data and optionally can use additional information in an ontology using *bindSchema* call.

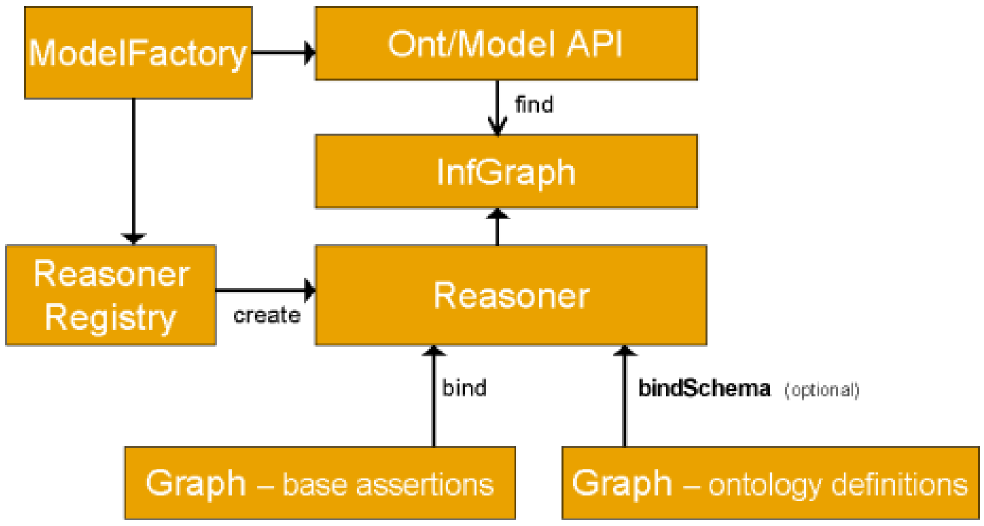


Figure 2.7. Jena inference architecture [29]

Jena includes the following reasoners in the distribution [29], where other DIG compliant reasoners can also be used safely:

1. Transitive reasoner: Provides support for storing and traversing class and property lattices. This implements just the *transitive* and *symmetric* properties of *rdfs:subPropertyOf* and *rdfs:subClassOf*.
2. RDFS rule reasoner: Implements a configurable subset of the RDFS entailments.
3. OWL, OWL Mini, OWL Micro Reasoners: A set of useful but incomplete implementation of the OWL/Lite subset of the OWL/Full language.
4. DAML micro reasoner: Used internally to enable the legacy DAML API to provide minimal (RDFS scale) inferencing.

5. Generic rule reasoner: A rule based reasoner that supports user defined rules. Forward chaining, tabled backward chaining and hybrid execution strategies are supported.

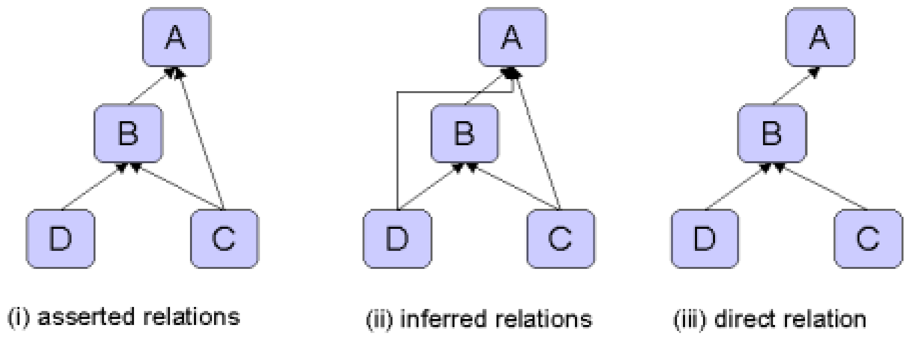


Figure 2.8. Jena transitive inference support [29]

In Jena, if a relation is transitive as in owl:subClassOf, then the relation can be specified as direct or inferred, which is the default behavior. However, in some cases it might be desirable to only infer direct relations. Jena supports the flexibility to reason in both ways.

2.4.2. OWL-S API

OWL-S API provides a Java API to read, write and execute OWL-S service descriptions. The API supports different versions of OWL-S (OWL-S 1.0, OWL-S 0.9, DAML-S 0.7) descriptions. OWL-S API also provides an execution engine that can invoke atomic processes that has WSDL and composite processes that uses control constructs Sequence, Unordered, and Split [31].

OWLReader and *OWLWriter* interfaces are provided for reading and writing OWL document respectively. *OWLOntology* object stores the read ontology content and it supports serialization of the ontology to a file through the use of *OWLWriter* interface.

OWL-S API provides methods that can directly read an OWL-S document to an internal semantic service representation. *OWLKnowledgeBase* construct has methods r*eadService(URI)* and *readAllServices(URI)* that assist reading services found at a certain URI [32]. Table 2.6 demonstrates the use of OWL-S API to read and write a service defined in OWL-S.

Table 2.6. Example demonstrating use of OWL-S API [32]

```
// create the URI of a known service for us to read in
URI aURI = URI.create("http://www.mindswap.org/2004/owl-s/1.1/ZipCodeFinder.owl");

// create a reader using the owl factory
OWLReader aReader = OWLFactory.createReader();

// read in the file specified by the URI using the newly created reader
OWLOntology aOntology = aReader.read(aURI);

// create an output stream to write the ontology to
FileOutputStream aOutputStream = new FileOutputStream("write-test.owl");

// create a writer using the owl factory
OWLWriter aWriter = OWLFactory.createWriter();

// write this ontology out to the specified output stream
aWriter.write(aOntology,aOutputStream);

// alternatively, we can create a KB object and use that to read in
// a file or set of files

// create a kb using the factory
OWLKnowledgeBase aKB = OWLFactory.createKB();

// now that we have a kb, we can read in the same file using the
// read method of the KB.  the read method of the kb will create an
// OWLReader similar to the above example to read in the file.  If you
// are reading in an OWL-S description you can use readService(URI)
// or readAllServices(URI) to get the service or list of services specfied
// by the URI.
Service aService = aKB.readService(aURI);
```

OWL-S API also provides a validation component named *OWLSValidator,* which takes the URI to an ontology, or a *OWLKnowledgeBase* object to validate [32].

2.4.3. WordNet

WordNet is an online lexical database for English words [62]. The development of WordNet began in 1985 at the Cognitive Science Laboratory of Princeton University. WordNet organizes words (nouns, verbs, adjectives and adverbs) into sets of synonyms called synsets. Semantic relations between synsets are provided with links. These relations are [62]:

1. *Synonymy* is WordNet's basic relation, because WordNet uses sets of synonyms (*synsets*) to represent word senses. Synonymy (*syn* same, *onyma* name) is a symmetric relation between word forms.
2. *Antonymy* (opposing-name) is also a symmetric semantic relation between word forms, especially important in organizing the meanings of adjectives and adverbs.
3. *Hyponymy* (sub-name) and its inverse, *hypernymy* (super-name), are transitive relations between synsets. Because there is usually only one hypernym, this semantic relation organizes the meanings of nouns into a hierarchical structure.
4. *Meronymy* (part-name) and its inverse, *holonymy* (whole-name), are complex semantic relations. WordNet distinguishes *component* parts, *substantive* parts, and *member* parts.
5. *Troponymy* (manner-name) is for verbs what hyponymy is for nouns, although the resulting hierarchies are much shallower.
6. *Entailment* relations between verbs are also coded in WordNet.

WordNet also provides the number of synsets that a certain word is contained in. The *frequency score* measure is provided to represent how often a word appears in a specific sense. WordNet organizes nouns and a verb into hierarchies of *is-a* relations [63]. WordNet also provides relations beyond *is-a*, including *has-part*, *is-made-of* and *is-attribute-of*. In addition to these, each concept is defined by a short gloss, which may contain an example use of word.

The hypernym/ hyponym relationships among the noun synsets enables WordNet to be interpreted as lexical ontology [63]. Considering WordNet as an ontology, it provides a basis to evaluate similarity between words. Several work has been performed on assessing similarity of words considering semantic relations. The next section introduces an open-source project, which discusses word similarity assessment approaches using WordNet database.

2.4.4. WordNet::Similarity

WordNet::Similarity is open-source software which makes it possible to measure the semantic similarity and relatedness between a pair of concepts (or synsets) [63]. It provides six measures of similarity and three measures of relatedness. The similarity measures are [64]:

1. Resnik: Depends on the idea that the similarity of two concepts is proportional to the information they share in common.

$$Sim_{res}(c_1,c_2) = IC(lsc(c_1,c_2)) \qquad (2.1)$$

   In equation 2.1, IC means information content, which is simply a measure of the specificity of a concept. And lcs(c1,c2) means the lowest common subsumer (LCS) of the two concepts. Resnik can be classified as a coarse measure, as all pairs of synsets that have the same LCS will have the same score.

2. Lin: As seen in equation 2.2, Lin measure augment the information content of LCS with the sum of the individual information contents of concepts.

$$Sim_{lim}(c_1,c_2) = 2*IC(lsc(c_1,c_2))/(IC(c_1)+IC(c_2)) \qquad (2.2)$$

3. Jcn: Jcn takes the difference of the sum of information contents of concepts and the information content of the LCS.

$$Sim_{jcn}(c_1,c_2) = 1/ (IC(c_1)+IC(c_2)-2* IC(lsc(c_1,c_2))) \quad (2.3)$$

4. Ich: Leacock and Chodorow rely on the length len(c1,c2) of the shortest path between two Synsets for their measure of similarity. However, they limit their attention to is–a links and scale the path length by the overall depth D of the taxonomy.

$$Sim_{lch}(c_1,c_2) = log(2*D/len(c_1,c_2)) \quad (2.4)$$

5. Wup: Wu & Palmer measure calculates similarity by considering the depths of the two Synsets in the WordNet taxonomies, along with the depth of the LCS.

$$Sim_{wup}(c_1,c_2) = log(2*depth(lsc(c_1,c_2))/depth(c_1)+depth(c_2)) \quad (2.5)$$

6. Random: It just uses random numbers as a measure of semantic similarity of word senses. *Maxrand* is the maximum length found in the *is–a* hierarchy in which concepts occur.

Measures of relatedness are more general and they are not constrained to *is-a* relations [63]:

1. Hso: Measures classifies relations in WordNet as having direction, and then establishes the relatedness between two concepts A and B by finding a path that is neither too long nor that changes direction too often.
2. Lesk: Finds overlaps between the glosses of concepts A and B, as well as concepts that are directly linked to A and B.
3. Vector: Creates a co–occurrence matrix for each word used in theWordNet glosses from a given corpus, and then represents each gloss/concept with a vector that is the average of these co–occurrence vectors.

WordNet::Similarity is implemented with Perl and provides modules that implement the above described similarity and relatedness approaches. It also exposes a Web interface to be used over HTTP.

2.5. Graph Matching

Graphs are powerful tools useful in various subfields of science and engineering. When graphs are used for the representation of structured objects, then the problem of measuring object similarity turns into the problem of computing the similarity of graphs, which is also known as graph matching [28].

In this study, graph matching is used for finding parameter matching in service and request pairs. Input and output parameters of services are represented as vertices in the graph and the matching between them are represented as edges.

2.5.1. Graph Matching Terminology

Graph algorithms and matching theory uses a terminology when defining theorems some of which are described below:

1. A vertex is matched if it is incident to an edge in the matching. Otherwise the vertex is unmatched.
2. A maximum matching is a matching that contains the largest possible number of edges.
3. A maximal matching is a matching M of a graph G with the property that if any edge not in M is added to M, it is no longer a matching. Every maximum matching must be maximal, but not every maximal matching must be maximum.
4. A perfect (complete) matching is a matching which covers all vertices of the graph. Every perfect matching is both maximum and maximal.
5. An alternating path is a path in which the edges belong alternatively to the matching and not to the matching.

6. An augmenting path is an alternating path that starts from and ends on free (unmatched) vertices.

2.5.2. Bipartite Graph Matching

Matching problem is mostly associated with bipartite graphs. Bipartite graphs are a special class of graphs where the set of vertices can be separated in two disjoint sets *U* and *V* such that no edge has both end-points in the same set.

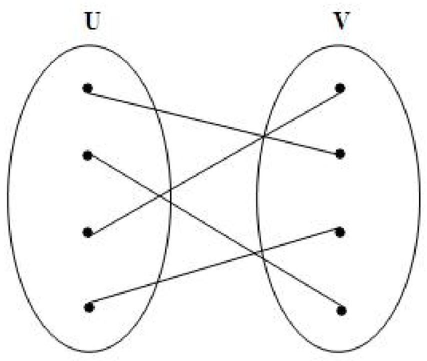


Figure 2.9. Bipartite graph

As mentioned before, in our matchmaker, service and request parameters can be represented as vertices of two disjoint sets. Each vertex is connected to every other vertex in the other set with a weight assigned on the edge. The problem turns into finding the subgraph, where each vertex is used only once and the sum of weights on the edges of subgraph is maximum. This problem is called maximum weight bipartite matching.

The problem of maximum bipartite matching, deals with pairing the vertices in the disjoint sets. The augmenting path algorithm finds a solution by adding an edge to the matching iterating over the elements of a single set. In a weighted bipartite graph, the problem turns into finding a matching where the sum of the values of edges is maximal. The problem is also known as the assignment problem.

A set of algorithms have been proposed to deal with the matching problem in bipartite graphs. In [9] the author lists them as displayed in figures 2.10 and 2.11:

Sequential Algorithms			
Date	Authors	Ref.	Complexity
	Primitive		$O(mn)$
1973	Hopcroft and Karp	[26]	$O(mn^{1/2})$
1991	Alt, Blum, Mehlhorn and Paul	[3]	$O(n^{3/2}\sqrt{m/\log n})$

Parallel Algorithms				
Date	Authors	Ref.	Complexity	Processors
1987	Kim and Chwa	[27]	$O(n \log n \log\log n)$	$O(n^3/\log n)$
1988	Goldberg, Plotkin and Vaidya	[24]	$O(n^{2/3}\log^3 n)$	$O(n^3/\log n)$
1992	Goldberg, Plotkin, Shmoys and Tardos	[23]	$O(m^{1/2}\log^3 n)$	$O(m^3)$

Figure 2.10. Non-weighted bipartite graphs [9]

Sequential Algorithms			
Date	Authors	Ref.	Complexity
1955	Kuhn	[29]	$O(mn^2)$
1972	Edmonds and Karp	[13]	$O(mn \log n)$
1984	Fredman and Tarjan	[13, 16]	$O(mn + n^2 \log n)$
1985	Gabow	[19]	$O(mn^{3/4}\log U)$
1989	Gabow and Tarjan	[21]	$O(mn^{1/2}\log nU)$

Parallel Algorithms				
Date	Authors	Ref.	Complexity	Processors
1988	Goldberg, Plotkin and Vaidya	[24]	$O(n^{2/3}\log^3 n \log nU)$	$O(n^3/\log n)$
1988	Gabow and Tarjan	[20]	$O((mn^{1/2}/p)\log p \log nU)$	$O(m/(n^{1/2}\log^2 n))$

Figure 2.11. Weighted bipartite graphs [9]

The primitive (augmenting path) algorithm applies a breadth-first search to find an M-augmenting path in the following way [9]:

1. Begin the search from M-free vertices in one bipartition of G, say V^+.
2. In odd iterations of search use edges that are not in M and in even iterations use M-edges

3. If the search finds an M-free vertex in V^-, then we have an M-augmenting path.

The search time for the above algorithm is bounded with O(mn). If the search finishes without finding an M-free vertex in V^-, then there is no M-augmenting path in G, therefore M is maximum.

Hungarian algorithm invented and published by Harold Kuhn in 1955, works on weighted bipartite graphs to find a minimum cost maximum matching [9]. The algorithm can also be used to find a maximum cost matching by reversing the cost function. Hungarian algorithm seeks to find a solution for the assignment problem and models it with a nxm cost matrix, where each element of the matrix represents the cost associated with assigning a certain job to the specified resource. This algorithm describes to the manual manipulation of a two-dimensional matrix by starring and priming zeros and by covering and uncovering rows and columns [72]. The following steps are described by the algorithm [72]:

1. Create an n x m matrix called the cost matrix in which each element represents the cost of assigning one of n workers to one of m jobs. Rotate the matrix so that there are at least as many rows as columns and let k=min(n, m).
2. For each row of the matrix, find the smallest element and subtract it from every element in its row. Go to step 3.
3. Find a zero (Z) in the resulting matrix. If there is no starred zero in its row or column, star Z. Repeat for each element in the matrix. Go to step 4.
4. Cover each column containing a starred zero. If K columns are covered, the starred zeros describe a complete set of unique assignments. In this case, go to 8, otherwise, Go to step 5.
5. Find a non-covered zero and prime it. If there is no starred zero in the row containing this primed zero, Go to step 6. Otherwise, cover this row and uncover the column containing the starred zero. Continue in this manner until there are no uncovered zeros left. Save the smallest uncovered value and go to step 7.
6. Construct a series of alternating primed and starred zeros as follows. Let Z_0 represent the uncovered primed zero found in step 5. Let Z_1 denote the starred zero in the column of Z_0 (if any). Let Z_2 denote the primed zero in the row of Z_1 (there

will always be one). Continue until the series terminates at a primed zero that has no starred zero in its column. Unstar each starred zero of the series, star each primed zero of the series, erase all primes and uncover every line in the matrix. Return to step 4.

7. Add the value found in step 5 to every element of each covered row, and subtract it from every element of each uncovered column. Return to step 5 without altering any stars, primes, or covered lines.
8. This step indicates the algorithm is completed. Assignment pairs are indicated by the positions of the starred zeros in the cost matrix. If C(i, j) is a starred zero, then the element associated with row i is assigned to the element associated with column j.

Examining the steps more carefully, in step 5, for example, the possible situations are, that there is a non-covered zero which gets primed and if there is no starred zero in its row the program goes onto step 6. The other possible way out of step 5 is that there are no non-covered zeros at all, in which case the program goes to step 7 [72].

3. Related Work

Semantic Web services aim to realize the vision of the Semantic Web, i.e. turning the Internet from an information repository for human consumption into a worldwide system for distributed Web computing [6]. The system is a machine-understandable media where all the data is combined with semantic metadata. The domain level formalizations of concepts form up the main element within this system, which is called ontology [11]. Ontology represents concepts and relations between the concepts; these can be hierarchical relations, whole-part relations, or any other meaningful type of linkage between the concepts [4].

The semantic matchmaking process is based on ontology formalizations over domains. In the upcoming section we present some of the selective research on the matchmaking process considering the concepts that we build our research on. Matchmaking of Web services considers the relationship between two services. The first one is called the advertisement and the other is called the request [2]. Advertisement denotes the services description of the existing services while the request indicates the picture of service requirements [19].

Traditional approaches to modeling semantic similarity between Web Services compute subsume relationship for function parameters in service profiles within a single ontology. In [20] a graph theoretic framework based on bipartite graph matching for finding the best correspondences among function parameters belonging to advertisement and request is introduced. On computing semantic similarity between a pair of function parameters, a similarity function is introduced, determining similar entity, which relaxes the requirement of a single ontology and accounts for the different ontology specifications. The function presented for semantic similarity across different ontologies provides an approach to detect similar parameters. The method is based on a matching process over weighted sum of synonym sets, semantic neighborhood, and distinguishing features. They make use of WordNet to capture the similarity between parameter names and also consider properties of concepts in matchmaking as distinguishing features. However, the method lacks use of contextual information or user preferences in matchmaking process.

In [22], a semantic ranking MSC is designed to rank the results of advertisements matchmaking. MSC stands for the initials of three factors' second words: Semantic Matching Degree (to capture the semantic aspects of attributes), Semantic Support (to describe the interestingness or potential usefulness of an attribute) and Relational Confidence (to capture the association relationships among attributes). Three categories of attributes are defined in advertisements matchmaking: *Generalizable Nominal Attribute (GNA)* whose values can form a concept hierarchy; *Numeric Attribute (NUA)*, called quantitative attribute, whose values are numeral; Nominal Attribute (NOA) whose values are neither numeral nor can form a concept hierarchy. However, in this study, the presented factors excluding Semantic Matching Degree, are not directly dealing with assessing service similarity to a certain request.

In [65], the authors introduce a step-by-step matchmaking process. They consider profile, input-output and non-functional attributes matching in the process. They also provide ranking of services according to their similarity to a certain request. However, they consider discrete scales of similarity measures: Exact, plug-in, subsume, intersection and fail. Although they provide ranking, they apply subsumption based reasoning and do not consider properties of concepts and contextual information in matchmaking. In [66], the authors describe capability matchmaking as matching inputs, outputs, preconditions and effects of services. They focus on the input-output matching. However, only a discrete scale classification of matching is provided. Besides, contextual information and properties of OWL concepts are not taken into account.

In [55], authors approach the problem of service similarity in a category-based approach. They define the following categories: Lexical similarity, which considers the textual similarity of service names; attribute similarity, which evaluates semantic similarity between Web service attributes and interface similarity, which considers the input-output parameter type and name similarity. They define a conceptual model of Web services, where a Web service is identified with its attributes and operations. Their study mostly focuses on the attribute similarity and focus on QoS attributes. The input-output parameter type similarity is only performed by a parameter type mapping table. The properties for those types defined in ontology are not considered. Besides, contextual information or user preferences are not considered in the matching process.

4. Problem Statement

The current industry standard for service discovery is UDDI [53]. However, UDDI has some shortcomings in that it returns coarse results for a keyword-based search and more importantly it lack semantics. It is basically a framework that supports category-based search.

Semantic service discovery tries to solve the above problem by utilizing capability-based search mechanism [25]. Capability based search enables reasoning on service input, output parameters, preconditions and effects. Service discovery tries to find all services that satisfy a certain request whereas service matchmaking deals with the relationship between two services [26]. Service discovery needs support of service matchmaking process.

The problem that we focus on in this research is to find suitable Web services that satisfy a certain request by applying a matchmaking process. A conventional matchmaker as described by Paolucci et al. [27] would distinguish four different relations between two concepts: Exact, subsume, plug-in and fail. This matchmaker would also determine the matching degree of a service to the request by the lowest degree of match. Considering the parameters, if one of them fails to match with request parameters the service would be considered as a fail to match. However, this scale of discrimination may not be appropriate in certain situations. The experimental research so far has shown that simple subsumption based matchmaking is not sufficient to capture semantic similarity [1, 22, 54, 55].

A discrete scale matchmaking methodology may result in false negatives. Some of the services in the candidate set might be eliminated due to not fitting those discrete scales. However, semantic relation has many dimensions other than subsumption. Next section describes those dimensions that we take into account, namely the properties of concepts and explicit annotations on concept similarity. A complete system should also evaluate all the candidate services and rank them according to their semantic similarity to a given request. We believe that considering semantic relations in depth provides a better matchmaking process.

Contextual information is also important to a matchmaking process. A matchmaking approach considering only raw OWL ontology definitions would operate without considering context. Matchmaking should also consider user preferences to be able to fully capture semantic similarity upon a user request.

Service composition also benefits from semantic Web to support automatic composition. Compositions can be generated dynamically by utilizing semantic descriptions of services to organize them in a workflow. In most of the composition approaches services are added to the composition one by one [56, 57]. As each service is added a matchmaking step is needed to make sure that the service supports the IOPE (Input, output, precondition and effect) constraints of the workflow node. Besides, the selected service's IOPE affects the matchmaking process for the next workflow node. Thus, a rich matchmaking framework is needed to obtain successful compositions. Considering alternative services in a composition is crucial for evaluation. So, a feature-rich and ranking based matchmaker is crucial for a composition engine.

In this research, we aim to provide an efficient and accurate matchmaking algorithm using scoring and ranking based on similarity distance information, extended subsumption and property level similarity assessment in a general semantic Web service discovery framework.

5. Proposed Solution

In this research, we propose a method that enhances simple subsumption based matchmaking approaches. Our main motivation is to capture semantic similarity between services in a more efficient way and eliminate false negatives. We consider service input and output parameters and perform the matchmaking considering the I/O (Input/Output) interface of services. Our proposed solution uses decision modules that can be plugged in and out. We have implemented some of these modules to add semantic relatedness values onto existing subsumption-based procedures. Our proposed matchmaker agent, SAM – Semantic Advanced Matchmaker, mainly provides ranking and scoring based on concept similarity. We also introduce the use of similarity distance annotations in an OWL document. Similarity distance supports explicitly annotating concept similarity in a numerical fashion. These annotations might refer to user's view of ontology and the similarity degree of concepts according to the user. Similarity distance is actually a method to represent context information in matchmaking process as described in Section 4.

Figure 5.1 overviews the matchmaking process in SAM. We assume that a conventional service discovery is performed on a request and as a result we have a relevant service set to apply matchmaking on. SAM gets the relevant service set and the request as inputs. The output of SAM is a ranked list of relevant service set. The focus of this study is the methods and procedures that take place in Matchmaking box described in Figure 5.1.

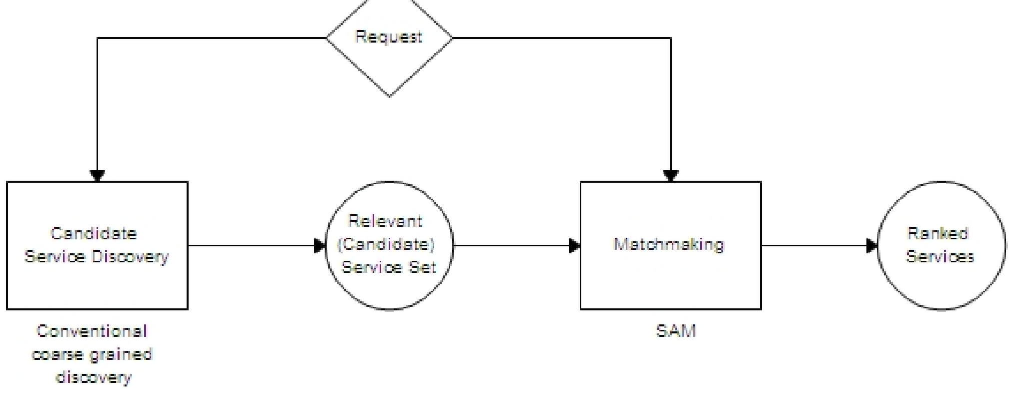

Request
Candidate Service Discovery
Relevant (Candidate) Service Set
Matchmaking
Ranked Services
Conventional coarse grained discovery
SAM

Figure 5.1. SAM matchmaking process

The components of the proposed architecture are shown in Figure 5.2. Request service definition and the corresponding relevant services set, which are discovered through conventional discovery mechanisms, are presented as input to the system.

Matchmaker component organizes the framework for comparing each of the services in the relevant service set with the request. It communicates with the Scoring Module to obtain similarity score for each parameter pair for a certain request-service pair. Finally, matchmaker component generates a bipartite graph for each request-service pair cooperating with Bipartite Graph Module. Each bipartite graph represents the service and request parameters with the similarity score assigned on every edge. As we are interested in retrieving best matching services compared to a request, bipartite graph matching algorithm will find the maximum weight match in each bipartite graph. This step ends up with parameter pairings in bipartite graphs representing maximum weight match in each service and request pair. At the end of the matchmaking process, the services in the relevant service set are ranked according to their scores in bipartite-graphs, which of course represent their semantic similarity to the request. SAM Bipartite Matcher module uses the Hungarian algorithm of Kuhn (1955), as improved and presented by Lawler (1976). The bipartite matcher engine finds maximum-weight matching in a complete bipartite graph.

Scoring Module is the part of the system where similarity between concepts is scored according to several criteria. We propose a plug-in architecture here, so that additional scoring modules can be plugged in or out. Currently, we implemented three scoring modules: Subsumption based scoring, similarity distance scoring and WordNet similarity scoring. Pellet reasoner is used in association with the Scoring Module for inference in ontology. Details on how these scoring modules work are described in the following sections.

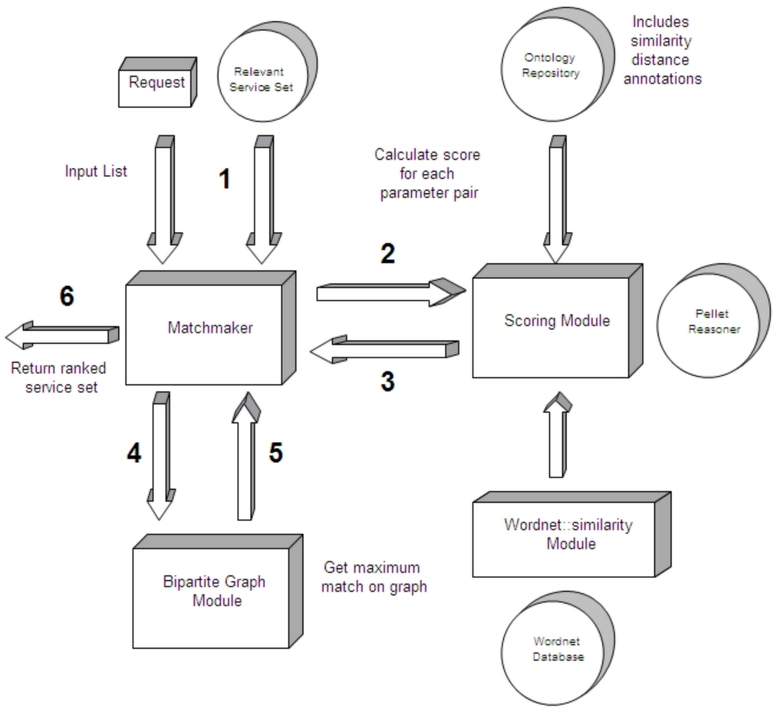

Request
Relevant Service Set
Ontology Repository
Includes similarity distance annotations
Input List
1
Calculate score for each parameter pair
2
6
Matchmaker
Scoring Module
Pellet Reasoner
Return ranked service set
3
4
5
Wordnet::similarity Module
Bipartite Graph Module
Get maximum match on graph
Wordnet Database

Figure 5.2. Matchmaking agent components

The main software components of our proposed matchmaking agent are shown in Figure 5.3. The top layer represents our matchmaker Semantic Advanced Matchmaker (SAM). OWL-S API models the service, profile, process and grounding ontologies of OWL-S in an easy to use manner. It is a widely used API in semantic applications. OWL-S API also presents interfaces for reasoning operations and utilizes Jena constructs at the back-end. At the bottom of the hierarchy we have Pellet reasoner for OWL reasoning operations.

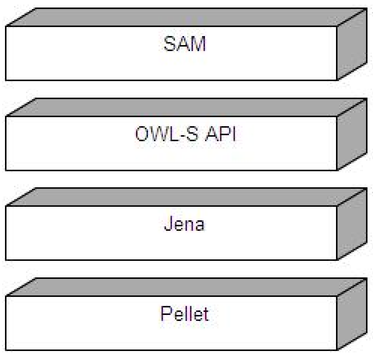


Figure 5.3. Software components of matchmaking agent

We believe that a discrete scale (exact, plug-in, subsume, and fail) of service classification is not sufficient for a matchmaking process. On the other hand, semantic ranking of services can capture a set of services that are lost in a discrete scale match. Semantic similarity assessment is a crucial step for the ranking process. In our proposed architecture, we present value-added similarity assessment approaches between service and request parameter pairs, which are described below.

5.1. Matching Algorithm

Previous research has shown that bipartite graph matching algorithm is a good fit for finding matching parameters in a service and request pair [9]. Bipartite graph matching provides us a solution for parameter pairing problem. We consider the inputs and outputs as separate cases and partition the service parameters and request parameters to form the bipartite graph. The similarity assessment process of our matchmaker assigns weights for each parameter pair on this bipartite graph. A maximum weight match on the final graph leaves us with the optimum matching parameter pairs and with a score that is sum of the weights between matched parameter pairs. We repeat this process for each service and request pair and finally rank the services according to their score from bipartite graph matching algorithm.

As we stated before the process that differentiates the services is the similarity assessment process. We consider OWL-S profiles of service definitions and assign similarity scores for input and output parameter pairs. We present the following value-added features for similarity assessment: Subsumption based similarity, property-level similarity, similarity distance information and WordNet similarity assessment.

It is possible that the input or output parameter count of request does not match the parameter count of the service. A preprocess step is introduced in the matchmaking process to add dummy parameters to the request or service in the following cases: If the service output parameter count is more than the request output parameter count an extra parameter is added to request outputs or if the request input parameter count is more than the service input parameter count an extra parameter is added to service inputs.

For the above cases, we introduced additional parameters to service or request to make the parameter count equal and to be able to support perfect bipartite matching. We ignored the additional output parameter after the matchmaking process. However, for the inputs we normalize the service similarity score. The idea is to apply a fair matchmaking so that a service that can provide the results with less input does not get a lower score due to its parameter count. So, we normalize the score of such a service using the following equation:

$$S_{new} = |P_r|/|P_s|*S_{old} \tag{5.1}$$

In equation 5.1, S_{old} represents the original service similarity score, $|P_r|$ and $|P_s|$ represent the parameter count of request and service, respectively. Finally, S_{new} represents the normalized service score.

An exception case is defined for the output parameters. If the service has less output parameters than the request, we do not directly eliminate the service with a zero score. We believe that any piece of provided information is valuable considering the outputs. So we classify the service as partially satisfying the request. The score for the matched output parameters are considered in the final evaluation. However, this is not the case for the input parameters. As a request with insufficient parameter set will not be able to invoke a certain service.

Another preprocess step in our matchmaking agent is to decompose any parameter type if it can be represented as a owl:unionOf element. This step ensures that all the parameters involved in matchmaking are atomic and reasoning on parameter count can be performed safely. As an example, let us suppose the concept “address” is expressed as a union of concepts “street” and “home”. If the request requires the parameters street and home as output and the service provides the parameter address, we may not capture the exact match between parameter pairs without such decomposition. The matchmaker could even return a failure to match as the service provides less output then required by the request.

After the decomposition of request and service parameters, the matchmaker returns a failure to match for the following cases: If the service has less output parameters then required by the request or if the request has less input parameters then required by the service.

After all the services in the candidate set are evaluated against the request, SAM stores the following information for each service and request pair: Match type (exact, subsume, plug-in, fail, property-level and ontology distance) for each matching parameter pair, similarity score (range between 0 and 1) for each matching parameter pair and total similarity score of service and request matching.

We provide ranking of services according to the above scores and in terms of their input and output parameters. However, considering both the input and output parameters we prioritize output matching as the outputs of a service are more important for client of a matchmaker. The following equation is used for this weighted calculation:

$$S_{final} = w_{input}*S_{inputs} + w_{output}*S_{outputs} \qquad (5.2)$$

In the above equation, S_{inputs} and $S_{outputs}$ represent the similarity score considering only the input parameters and output parameters respectively. w_{input} and w_{output} represents the weights for the input and output similarity scores, where they are fixed to 0.4 and 0.6 after several runs of matchmaker considering the expected outcome. Finally, S_{final} represents the final score of similarity considering both the input and output parameters.

$$S_{x,y} = w_{sub}*Subsumption_{x,y} + w_{word}*WordNet_{x,y} \qquad (5.3)$$

$S_{x,y,}$ in the above equation, represents final similarity score between concepts x and y. $Subsumption_{x,y}$ represents semantic score obtained through subsumption, property level matching and semantic distance. $WordNet_{x,y}$ represents the WordNet score for concept names. The coefficients for subsumption and WordNet are fixed at 0.9 and 0.1 after making experimental runs. We plan to apply a neural network training approach to determine values for coefficients utilizing a large training data in future.

As we have indicated before, a feature-rich matchmaking process is crucial for capturing semantic similarity between parameter pairs. Following sections describe the value-added approaches that our matchmaking agent applies in order to evaluate semantic similarity and rank the services according to the final similarity score.

5.1.1. Subsumption based Similarity Assessment

We make use of OWL-DL constructs *subClassOf, disjointWith, complementOf, unionOf and intersectionOf* to assess concept similarity based on subsumption. If two concepts are explicitly stated to be complement or disjoint, a zero score is directly assigned. Otherwise, we check for subconcept relation and also assess according to property level assessment procedure described below.

We wanted to capture similarity values in bipartite graph since it is important to decompose concepts that include the characteristic of “a union of”. Following this approach, we always pair and assess score for atomic concepts in matchmaking process.

Considering the input and output parameter matching, the following cases are favored in subsumption: Request input parameter subsumed by the service input parameter or service output parameter subsumed by the request output parameter.

The following equation explains the above scoring differentiation for subsumption relation in parameters:

$$S_{final} = w * S_{x,y} \tag{5.4}$$

$S_{x,y}$ represents the semantic similarity score between parameters x and y, where w represents the weight that is adjusted according to the subsumption relation. For the above described favored subsumption cases we assign 0.6 to w and 0.4 otherwise. S_{final} represents the final adjusted similarity score between the concepts.

5.1.2. Property-level Similarity Assessment

We believe that in matchmaking it is also important to have properties and their associated range in measuring the degree of match. Such as, if two concepts have similar properties (properties having subclass relation) and their range classes are similar, then this improves their level of similarity.

Our proposed matchmaker checks for property level similarity in a recursive fashion. A complete similarity assessment is performed for range classes of each matching property. This means that if the range classes of similar properties have also common properties, property level matching is again applied to capture similarity at that level.

As the properties of a parent concept are inherited by its child concepts in a semantic network, property-level matching is crucial to capture semantic similarity between sibling concepts. Such concepts are not in a subsumption relation. However, since they have a common parent, it is highly likely that they contain common properties with similar range types.

As an example, considering the ontology in Figure 6.3, *Foreign-Magazine* and *ScienceFictionBook* concepts are not in a subsumption relation. A conventional matchmaker would classify these concepts as non-related. However, these concepts have properties *datePublished* and *publishedBy* in common, which refer to concepts *Date* and *Publisher* respectively as ranges. This means both concepts have a publisher and a publishing date, what makes them similar at a certain degree. In order to calculate the degree of match, SAM evaluates the similarity of range objects for *datePublished* and *publishedBy* properties associated with concepts *Foreign-Magazine* and *ScienceFictionBook.* In this example ontology, *Foreign-Magazine* has range *Date* for *datePublished* and *Premium-Publisher* for *publishedBy* property. *ScienceFictionBook* has also range *Date* for *datePublished* and *Alternative-Publisher* for *publishedBy* property. As a result, the similarity degree of concepts *Date* compared to *Date* and *Premium-Publisher* compared to *Alternative-Publisher*, gives an idea on how much the concepts *Foreign-Magazine* and *ScienceFictionBook* are similar.

The following equation describes how property-level matching is used in scoring:

$$Sp_{x,y} = w_p * \sum S_{m,n} \qquad (5.5)$$

In the above equation x and y represents the concepts that have common properties. The property-level matching score for the concepts x and y is expressed as $Sp_{x,y.}$ Let us suppose m and n represent associated range classes for each matching property of x and y. $S_{m,n}$ represents the similarity score between concepts m and n. w_p represents the weight for property-level matching affecting the total similarity score, where we defined it to be equal to 0.1 after several runs. Then, $Sp_{x,y}$ equals to the adjusted sum of similarity scores for each m and n. Note that SAM considers object type properties for subsumption. For data type properties similarity will be simply equal to 0 or 1.

Using property level similarity assessment ranks a service that would normally be eliminated by a conventional matchmaker. For example, a user request may favour a particular author for a novel. A service, which returns articles that are written by that particular author, will have a high score even though the concept of “an article” does not compare to the concept of “a novel”. Therefore our proposed architecture returns positive results for concepts that have similar properties as well as the similar concepts.

5.1.3. Similarity Distance based Assessment

Similarity distance represents the semantic similarity between two concepts in terms of a value in the range as [0,1]. It is an explicit annotation on the ontology, where contextual information is represented. Similarity distance annotations can be considered as a user’s point of view on semantic relations of concepts or similarity relations of concepts in a certain context.

Similarity distance annotations can be introduced into ontology by the ontology creators or by the user’s of a system dynamically. A system can provide the user with an interface to assess similarity relations between concepts in a user-friendly way. The user can classify concepts as being equal, very similar, similar or not related so that the system can map these relation classes to the range [0,1]. Our implementation does not focus on the semantic distance annotation step and assumes that the ontology with semantic distance annotation is given.

In [24], authors make use of semantic distance information by representing annotations in a custom XML file accompanying the ontology document. We further improve this approach and apply a standards based approach. We defined a similarity distance ontology which represents how similarity values between concepts are defined. Importing similarity distance ontology and annotating the concepts with similarity distance concept will enable any ontology to represent similarity relation between its concepts. This way we provide a standard representation and enable an ontology which has similarity distance annotations to be processed in any OWL processor.

To represent similarity distance information we applied N-ary relation pattern in OWL, which is used to represent additional attributes on a property [58]. The additional attribute in our case is the similarity distance value. Figure 5.4 shows how this pattern is organized:

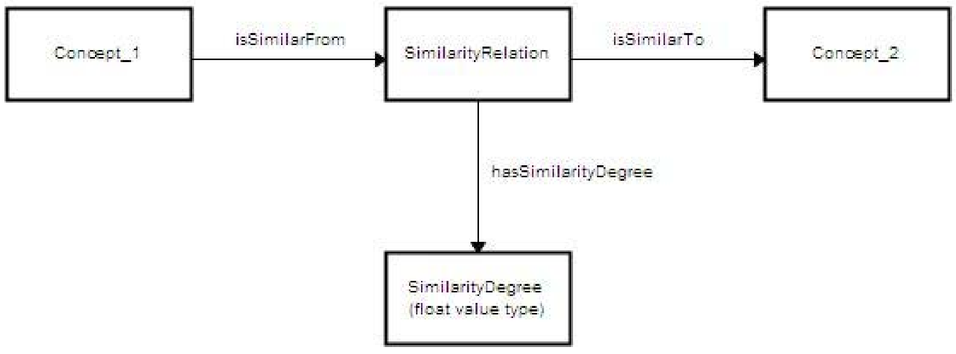


Figure 5.4. N-ary relation pattern in OWL, representing similarity distance information

SimilarityRelation concept is introduced as a class with this pattern and the similarity distance value is represented as the range of *hasSimilarityDegree* property of this concept. The similar classes are represented as *Concept_1* and *Concept_2* in Figure 5.4. The relations *isSimilarFrom* and *isSimilarTo* attach the concepts to the similarity relation and indeed they are symmetric. Similarity distance ontology is represented below:

Table 5.1. OWL representation of similarity distance ontology

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns="http://127.0.0.1/ontology/similarityOntology.owl#"
  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#"
  xml:base="http://127.0.0.1/ontology/similarityOntology.owl">
  <owl:Ontology rdf:about="http://127.0.0.1/ontology/similarityOntology.owl"/>
    <owl:Class rdf:ID="SimilarityRelation">
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:hasValuerdf:datatype="http://www.w3.org/2001/XMLSchema#float">
          -1</owl:hasValue>
          <owl:onProperty>
            <owl:DatatypeProperty rdf:ID="hasSimilarityDistance"/>
          </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:allValuesFrom rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="isSimilarFrom"/>
          </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:allValuesFrom rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="isSimilarTo"/>
          </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
  </owl:Class>
  <owl:ObjectProperty rdf:about="#isSimilarFrom">
    <rdfs:domain rdf:resource="#SimilarityRelation"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#isSimilarTo">
    <rdfs:domain rdf:resource="#SimilarityRelation"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:about="#hasSimilarityDistance">
    <rdfs:domain rdf:resource="#SimilarityRelation"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

The similarity distance formulation is defined as follows:

$$Sd_{x,y} = Sd_{x,t} * Sd_{t,k} * \ldots * Sd_{m,y} \tag{5.6}$$

In the above equation, $Sd_{a,b} \in [0,1]$ for any a and b pair. $Sd_{x,y}$ represents similarity distance between concepts x and y. The product of similarity distance values on the path from x and y gives the value for $Sd_{x,y}$. If the concepts are not subclasses of each other then we take the path including their first common parent in the hierarchy. If there are more than one path between two concepts (occurs if a concept has more than one parent) we take the path with higher score.

Similarity distance assignment approach is not strictly defined. Indeed, we encourage similarity assignment to be performed in a consistent way, considering the ontology as a whole. In addition, if similarity distance annotation is not found between two concepts, then a default distance value is assigned according to the following equation:

$$Sd_{x,y} = 1/|subClassOf(x)_{direct}| \tag{5.7}$$

In the above equation $Sd_{x,y}$ represents similarity distance between concepts x and y and $|subClassOf(x)_{direct}|$ represents the number of elements in set of direct subclasses of concept x.

5.1.4. WordNet based Similarity Assessment

As described in section 2, *WordNet* organizes concepts in synonym sets and provide information on semantic relations between synsets by making use of pointers. In our architecture we take *WordNet* as a secondary source of information in addition to the ontology repository. We aimed at reasoning with these highly structured information sources in order to get more reliable result sets.

We make use of the previously described *Wordnet::Similarity* source project to assess similarity score among words. The path length criterion is used for score assignment. The parameter types of services are presented as input to *Wordnet::Similarity* module.

SAM provides *WordNet* scoring as a module that can be turned on and off. As we present parameter type names extracted from service ontologies as input to *Wordnet::Similarity*, some types may not correspond to any word name in *WordNet*. Ontology designers may not provide concept names that correspond to valid words in *WordNet* database. Considering this, some of the concepts might have zero score from *WordNet* in matchmaking process according to equation 5.3, while others gain some considerable score. Therefore, it might be desirable not to consider *WordNet* scoring in some matchmaking scenarios. SAM supports this flexibility.

6. Experimental Results

6.1. Test Ontology

The ontology and services we used are retrieved from "OWL-S Service Retrieval Test Collection version 2.1". The services in the collection are mostly extracted from public UDDI registries, providing 582 Web services described in OWL-S and from seven different domains. The OWL-S Test collection version 2.1 contains 29 queries, each of which associated with a set of 10 to 15 services [8].

In order to evaluate the performance of our proposed matchmaking agent we extended the book ontology in OWL-S Service Retrieval Test Collection (OWL-S TC) and also modified related request and service definitions accordingly [8]. As shown in Figure 6.1, we added subclasses of *Magazine*, namely *Foreign-Magazine* and *Local-Magazine* classes. We introduced subclasses for *Publisher* concept, *Author* and *Newspaper* concepts. We also annotated the book ontology with similarity distance information, making use of similarity distance ontology that we have imported.

The ontology contains information on printed material classification and related concepts such as authors, publishers, publishing intervals in terms of time and date and several other concepts. Figures 6.1 to 6.3 are sections from the book ontology.

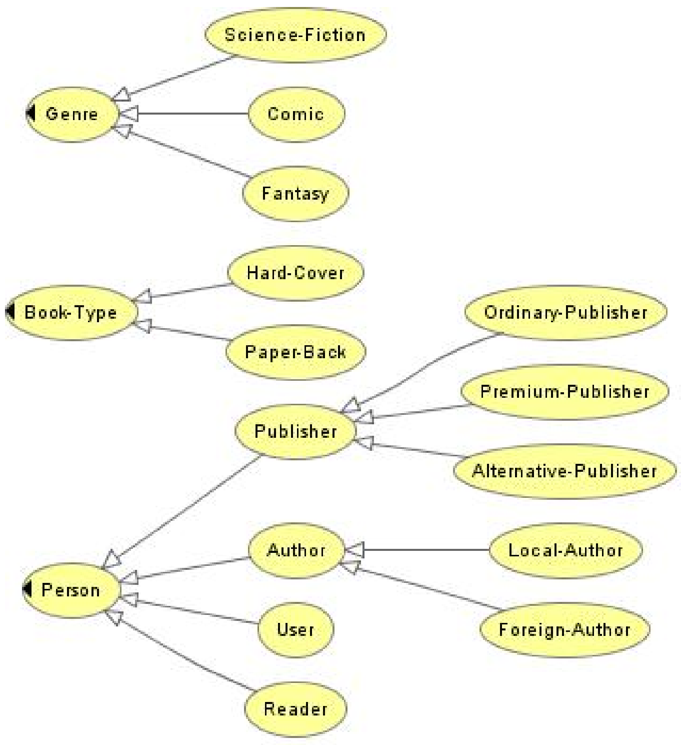


Figure 6.1. A section of book ontology

As shown above we introduced subclasses for Publisher and Author concepts. This will provide further differentiation in matchmaking process when considering the above concepts in parameter types.

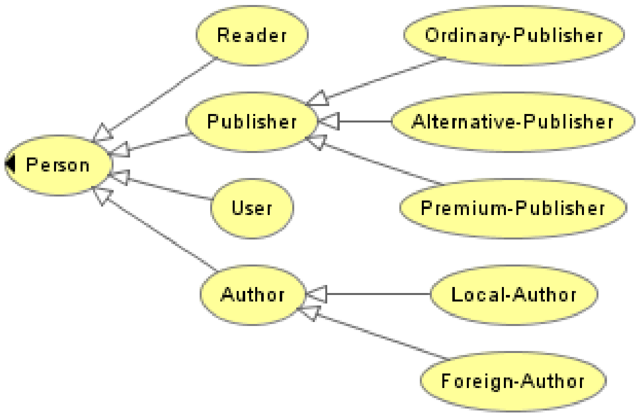


Figure 6.2. Person ontology section

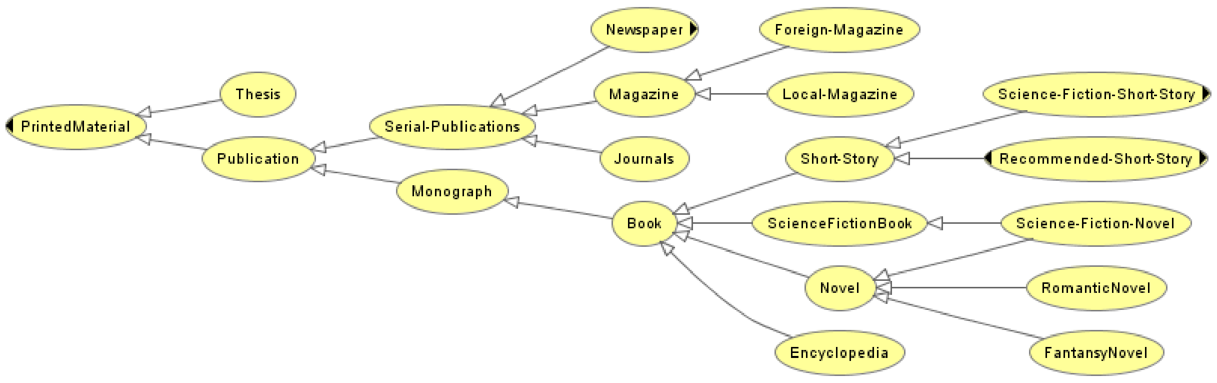


Figure 6.3. Printed Material ontology section

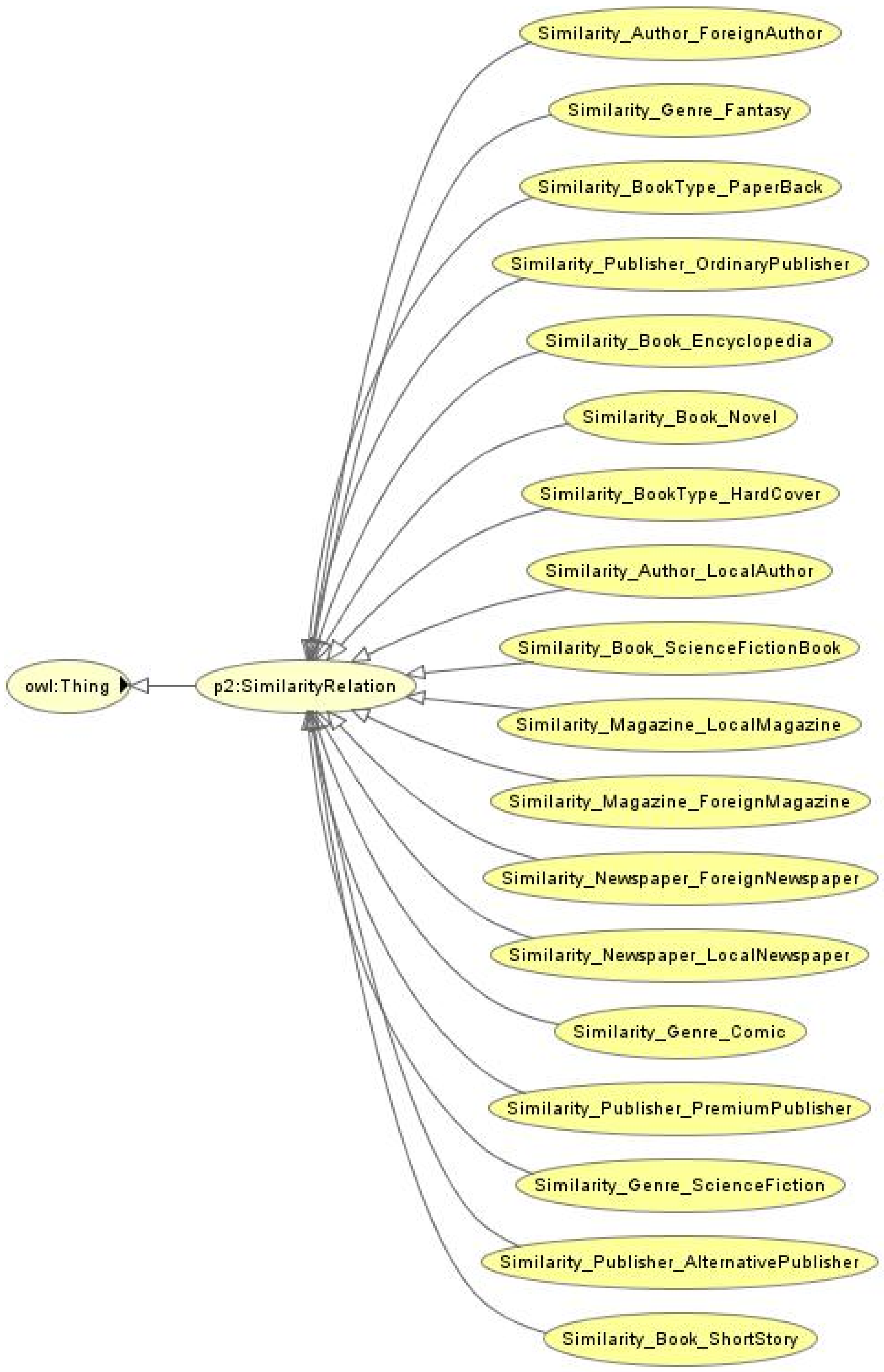

Similarity_Author_ForeignAuthor
Similarity_Genre_Fantasy
Similarity_BookType_PaperBack
Similarity_Publisher_OrdinaryPublisher
Similarity_Book_Encyclopedia
Similarity_Book_Novel
Similarity_BookType_HardCover
Similarity_Author_LocalAuthor
Similarity_Book_ScienceFictionBook
owl:Thing
p2:SimilarityRelation
Similarity_Magazine_LocalMagazine
Similarity_Magazine_ForeignMagazine
Similarity_Newspaper_ForeignNewspaper
Similarity_Newspaper_LocalNewspaper
Similarity_Genre_Comic
Similarity_Publisher_PremiumPublisher
Similarity_Genre_ScienceFiction
Similarity_Publisher_AlternativePublisher
Similarity_Book_ShortStory

Figure 6.4. SimilarityRelation concept and subclasses

As shown in Figure 6.4, we imported similarity distance ontology and introduced subclasses of SimilarityRelation concept to represent similarity distance annotations. This way, we can represent semantic similarity values between book ontology concepts. Following table lists the similarity distance values assigned in the book ontology:

Table 6.1. Similarity distance values in book ontology

Superclass	**Subclass**	**Similarity Distance**
Publisher	Ordinary-Publisher	0.2
Publisher	Alternative-Publisher	0.5
Publisher	Premium-Publisher	0.3
Author	Local-Author	0.3
Author	Foreign-Author	0.7
Magazine	Local-Magazine	0.3
Magazine	Foreign-Magazine	0.7
Newspaper	Local-Newspaper	0.3
Newspaper	Foreign-Newspaper	0.7
Book	Short-Story	0.7
Book	Science-Fiction-Book	0.4
Book	Novel	0.3
Book	Encyclopedia	0.1
Novel	Science-Fiction-Novel	0.6
Novel	Fantasy-Novel	0.2
Novel	Romantic-Novel	0.2
Book-Type	Hard-Cover	0.7
Book-Type	Paper-Back	0.3
Genre	Comic	0.3
Genre	Fantasy	0.2
Genre	Science-Fiction	0.5

The above similarity distance values are annotated in the ontology as described in Table 6.2 below. The similarity distance value in super concept *SimilarityRelation* is overridden by the annotation. *Owl:hasValue* construct is used to represent the similarity distance. The concepts that are described as similar are represented with *isSimilarFrom* and *isSimilarTo* properties using *owl:Restriction* construct.

Table 6.2. Similarity distance annotation

```
<owl:Class rdf:ID="Similarity_Author_LocalAuthor">
<rdfs:subClassOf
  rdf:resource="http://127.0.0.1/ontology/similarityOntology.owl#SimilarityRelation"/>
<rdfs:subClassOf>
   <owl:Restriction>
      <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#float"> 0.3
      </owl:hasValue>
      <owl:onProperty  rdf:resource=
      "http://127.0.0.1/ontology/similarityOntology.owl#hasSimilarityDistance"/>
   </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
   <owl:Restriction>
      <owl:onProperty rdf:resource=
       "http://127.0.0.1/ontology/similarityOntology.owl#isSimilarFrom"/>
      <owl:allValuesFrom rdf:resource="#Author"/>
   </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
   <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#Local-Author"/>
      </owl:allValuesFrom>
      <owl:onProperty rdf:resource=
       "http://127.0.0.1/ontology/similarityOntology.owl#isSimilarTo"/>
    </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

The OWL-S documents describing the services make use of the book and concepts ontology in OWL-S TC. Table 6.3 represents the header section of an OWL-S document demonstrating how these ontologies are imported.

Table 6.3. OWL-S document header

```
<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Service.owl" />
<owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Process.owl" />
<owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Profile.owl" />
<owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Grounding.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/modified_books.owl" />
<owl:imports rdf:resource="http://127.0.0.1/ontology/concept.owl" />
</owl:Ontology>
```

Below, Table 6.4 presents the service section of an OWL-S document.

Table 6.4. OWL-S document service section

```
<service:Service rdf:ID="BOOK_PRICE_SERVICE">
<service:presents rdf:resource="#BOOK_PRICE_PROFILE"/>
<service:describedBy rdf:resource="#BOOK_PRICE_PROCESS_MODEL"/>
<service:supports rdf:resource="#BOOK_PRICE_GROUNDING"/>
</service:Service>
```

Table 6.5 represents the profile section of an OWL-S document, describing how the inputs and outputs of the service are referred.

Table 6.5. OWL-S document profile section

```
<profile:Profile rdf:ID="BOOK_PRICE_PROFILE">
<service:isPresentedBy rdf:resource="#BOOK_PRICE_SERVICE"/>
<profile:serviceName xml:lang="en">
…
</profile:serviceName>
<profile:textDescription xml:lang="en">
…
</profile:textDescription>
<profile:hasInput  rdf:resource="#_ORDINARYPUBLISHER"/>
…
<profile:hasOutput rdf:resource="#_GENRE"/>
<profile:has_process rdf:resource="BOOK_PRICE_PROCESS" />
</profile:Profile>
```

Table 6.6 presents the process section of the OWL-S document and represents how types of input and output parameters are defined.

Table 6.6. OWL-S document process section

```
…
<process:AtomicProcess rdf:ID="BOOK_PRICE_PROCESS">
<profile:hasInput  rdf:resource="#_ORDINARYPUBLISHER"/>
…
<profile:hasOutput rdf:resource="#_GENRE"/>
</process:AtomicProcess>
…
<process:Input rdf:ID="_NOVEL">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://127.0.0.1/ontology/modified_books.owl#Novel
</process:parameterType>
</process:Input>
…
<process:Output  rdf:ID="_GENRE">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://127.0.0.1/ontology/modified_books.owl#Genre
</process:parameterType>
</process:Output>
```

6.2. Test Environment

SAM is developed in Java with Eclipse IDE. Java is a good choice considering its power in open source community and semantic Web projects that are developed in Java [29, 30, 31, 32]. As described in section 5, SAM makes use of OWL-S API for processing OWL-S documents and Jena for reasoning on OWL. Apache Web server is used to publish OWL ontology documents and services described with OWL-S. We also used Protégé ontology editor to edit and visualize OWL documents.

6.3. Test Results

In order to demonstrate the value-added features of our proposed matchmaker, we present two test cases for request-service matchmaking. The service and request pairs are described in Tables 6.7 and 6.8.

Table 6.7. Test case 1

Service Name	**Inputs**	**Outputs**
Request	Ordinary-Publisher, Novel, Paper-Back	Local-Author, Genre
Service 1	Publisher, ScienceFictionBook	Author, Price
Service 2	Book, Alternative-Publisher, Book-Type	Publisher, Price, Date
Service 3	FantasyNovel, Author	Price, Comic
Service 4	Newspaper, Book-Type, Person	Review, Fantasy
Service 5	Publication, Book-Type, Reader	Genre, Publisher

Table 6.8. Test case 2

Service Name	Inputs	Outputs
Request	ScienceFictionShortStory, Foreign-Magazine	Foreign-Author, Publisher
Service 1	Local-Author, Newspaper	Ordinary-Publisher, Price
Service 2	RecommendedShortStory, Foreign-Magazine	Author, Publisher
Service 3	Novel, Magazine	Reader, Author
Service 4	Local-Magazine	Ordinary-Publisher, Price, Reader

Results for input parameter matching for test case 1 are listed in Table 6.9.

Table 6.9. Input parameter matching for test case 1

Service Name	**Parameter Pairing**	**Service Score**
Service 1	*Novel to Science-Fiction-Book (Property Level) : 0.13176 *Ordinary-Publisher to Publisher (Request Subsumed) : 0.108	0.35964
Service 2	*Novel to Book (Request Subsumed + WordNet) : 0.266 *Paper-Back to Book-Type (Request Subsumed) : 0.162 *Ordinary-Publisher to Alternative-Publisher (Only Distance) : 0.0108	0.4388
Service 3	*Novel to Fantansy-Novel (Service Subsumed) : 0.12 *Ordinary-Publisher to Author (Only Distance) : 0.00018	0.18026
Service 4	*Novel to Newspaper (Property Level + WordNet) : 0.07225 *Paper-Back to Book-Type (Request Subsumed) : 0.162 *Ordinary-Publisher to Person (Request Subsumed) : 0.00211	0.23636
Service 5	*Novel to Publication (Request Subsumed + WordNet) : 0.155 *Paper-Back to Book-Type (Request Subsumed) : 0.162 *Ordinary-Publisher to Reader (Only Distance) : 0.00018	0.31718

Service 2 has the highest score considering input parameters only. The request input parameter *Novel* is a direct subclass of service parameter *Book*. For the input parameter matching, service parameter subsuming the request parameter is favored considering equation 5.4. An extra score of 0.5 is also received from WordNet similarity for *Novel* and *Book* parameters. We might not have considered WordNet score, considering that concept names such as *Paper-Back* or *Book-Type* are not included in WordNet database. Even if that was the case Service 2 still has the highest score. The same favorable case also holds for *Paper-Back* and *Book-Type* concepts as they have direct subclass relationship. The last parameter pair, *Ordinary-Publisher* and *Alternative-Publisher*, has no property in common and they are sibling concepts. Because of that we just consider their distance in ontology calculating with the similarity distance annotations according to equation 5.6.

Service 1 has the next highest score although it operates with only 2 input parameters. We normalized Service 1's score proportional to 3/2, according to equation 5.1. *Novel* and *Science-Fiction-Book* concepts are sibling nodes in ontology and they have the following common properties: *publishedBy*, *datePublished*, *timePublished*, *writtenBy*, *hasType* and *isTitled*. We apply property level matching and compare the semantic similarity for the range types of these common properties. The range types are exact matches in terms of similarity except *publishedBy* property. *Novel* has type *Publisher* and *Science-Fiction-Book* has type *Alternative-Publisher* respectively. So, a property-level similarity score for this subsumption relation is added to the parameter pair scores, considering equation 5.5. The other parameter pair, *Ordinary-Publisher* and *Publisher* has also subsumption relation in the favorable case, where the service parameter subsumes the request.

Service 5 also demonstrates parameter pairs having WordNet similarity, *Novel* and *Publication*. *Ordinary-Publisher* and *Reader* concepts both have the *Person* concept as parent. But they do not have a subsumption relation and we calculate their similarity score by considering the path in ontology between them and the similarity distance annotations on this path.

Novel and *Newspaper* concepts in Service 4, has *Publication* concept as common parent but they are on different paths in ontology. However, they have common properties and we consider those property ranges in terms of similarity and add to the final score.

Service 3 is also an example of score normalization in input parameter matching case. It has the weakest match among all services: A subsumption relation with an unfavored case, where request parameter subsumes the request and similarity distance score from *Ordinary-Publisher* to *Author*, where they do not have any properties in common.

The following table lists the results for output parameter matching for test case 1.

Table 6.10. Output parameter matching for test case 1

Service Name	**Parameter Pairing**	**Service Score**
Service 1	*Genre to Price (No Match + WordNet) : 0.01429 *Local-Author to Author (Request Subsumed) : 0.108	0.12229
Service 2	*Genre to Comic (Service Subsumed + WordNet) : 0.17033	0.17033
Service 3	*Genre to Price (No Match + WordNet) : 0.01429 *Local-Author to Publisher (Only Distance) : 0.00018	0.01447
Service 4	*Genre to Genre (Exact + WordNet) : 1.0 *Local-Author to Publisher (Only Distance) : 0.00018	1.00018
Service 5	*Genre to Fantasy (Service Subsumed + WordNet) : 0.12229	0.12229

Service 4 has a great score advantage considering the exact match for *Genre* parameter in outputs. In contrast to input parameter matching, request parameter subsuming the service parameter is favorable in subsumption relation for outputs. We have this property is Service 2 and Service 5.

We can see that Service 1 has no semantic similarity for *Genre* and *Price* considering the book ontology. However, we have a score of 0.01429 coming from WordNet database. For Service 1 and Service 5 we have the same output score and we do rank randomly for such cases.

Table 6.11. Final service rank for test case 1

Service Name	Service Score
Service 1	0.21723
Service 2	0.27771
Service 3	0.08078
Service 4	0.69465
Service 5	0.20024

Considering both the input and output matching with the equation 5.2, we have the above ranking in table 6.11. Service 4 has the advantage of exact match in output parameters. Service 2 has favorable similarity relations in input parameters and it is also the second in ranking for output parameters. Finally, Service 3 has the weakest match for both inputs and outputs, which makes it the last in ranking.

To better demonstrate the advantages of using SAM as compared to a conventional matchmaker, we have also implemented a conventional matchmaker. This matchmaker does not have property-level similarity matchmaking or similarity distance annotation capabilities. So, if the concepts do not have a subsumption relation they will have a zero matchmaking score. If there is a subsumption relation then the following equation will apply in addition to equation 5.4:

$$S = 1 / D_{x,y} \tag{6.1}$$

In equation 6.1, S represents the subsumption similarity score, where D is the path length incremented by 1, between the concepts x and y in ontology. Finally, according to equation 5.4 this similarity score is multiplied by an adjustment factor for subsume or plug-in cases.

As shown in table 6.12, we have a loss of fine-grained discrimination between services. Although Service 1 had property-level matching in SAM, now it loses the advantage and ranked below Service 4 and Service 5. Service 4 and Service 5 are also equal in ranking due to similarity distance and property-level matching non-existence.

Table 6.12. Conventional matchmaker - Input parameter matching for test case 1

Service Name	**Parameter Pairing**	**Service Score**
Service 1	*Ordinary-Publisher to Publisher (Request Subsumed) : 0.54	0.81
Service 2	*Novel to Book (Request Subsumed) : 0.54 *Paper-Back to Book-Type (Request Subsumed) : 0.54	1.08
Service 3	*Novel to FantansyNovel (Service Subsumed) : 0.36	0.54
Service 4	*Paper-Back to Book-Type (Request Subsumed) : 0.54 *Ordinary-Publisher to Person (Request Subsumed) : 0.54	1.08
Service 5	*Novel to Publication (Request Subsumed) : 0.54 *Paper-Back to Book-Type (Request Subsumed) : 0.54	1.08

Table 6.13 demonstrates the output parameter matching with a conventional matchmaker. Although the difference is not as obvious as the case in inputs, we can observe that Service 1 has lost the advantage of WordNet score, which is a second source

of information in SAM. Service 3 also now gets a zero score as similarity distance and WordNet is not considered, which means it is totally left out in this match.

Table 6.13. Conventional matchmaker - Output parameter matching for test case 1

Service Name	**Parameter Pairing**	**Service Score**
Service 1	*Local-Author to Author (Request Subsumed) : 0.36	0.36
Service 2	*Genre to Comic (Service Subsumed) : 0.54	0.54
Service 3	No Match	0
Service 4	*Genre to Genre (Exact) : 0.9	0.9
Service 5	*Genre to Fantasy (Service Subsumed) : 0.54	0.54

Table 6.14. Conventional matchmaker - Final service rank for test case 1

Service Name	**Service Score**
Service 1	0.54
Service 2	0.756
Service 3	0.216
Service 4	0.972
Service 5	0.756

According to table 6.14 the final ranking changes for Service 5 and Service 1. However, as we have more test scenarios and more services in the relevant set, the difference will be more obvious and a fine-grained scoring as in SAM will be needed for a clear differentiation.

Table 6.15 lists the input parameter matching for test case 2. The exact match in parameter *Foreign-Magazine* ranks Service 2 top in input matching. Service 2 also has property level matching as *Science-Fiction-Short-Story* and *Recommended-Short-Story* are sibling nodes in ontology and share common properties. Service 4 has a property level matching for parameters *Foreign-Magazine* and *Local-Magazine* as they are sibling nodes in ontology. Besides, Service 4's score is adjusted with a ratio of 3/1 according to request input parameter count.

Table 6.15. Input parameter matching for test case 2

Service Name	**Parameter Pairing**	**Service Score**
Service 1	*Foreign-Magazine to Newspaper (Property Level) : 0.07065	0.07065
Service 2	*Foreign-Magazine to Foreign-Magazine (Exact) : 0.9 *Science-Fiction-Short-Story to Recommended-Short-Story (Property Level) : 0.19764	1.09764
Service 3	*Foreign-Magazine to Magazine (Request Subsumed) : 0.378 *Science-Fiction-Short-Story to Novel (Property Level) : 0.12587	0.50387
Service 4	*Foreign-Magazine to Local-Magazine (Property Level) : 0.12748	0.25495

As listed in table 6.16, Service 2 ranks top in output parameter matching due to an exact match with the *Publisher* parameter. Service 3 has both WordNet and similarity distance matching which puts it into second place in ranking. Service 4 has a slight advantage over Service 1 as it has the parameter *Reader* which shares a common parent with parameter *Foreign-Author* in ontology.

Table 6.16. Output parameter matching for test case 2

Service Name	Parameter Pairing	Service Score
Service 1	*Publisher to Ordinary-Publisher (Service Subsumed) : 0.108	0.108
Service 2	*Foreign-Author to Author (Request Subsumed) : 0.252 *Publisher to Publisher (Exact + WordNet) : 1.0	1.252
Service 3	*Foreign-Author to Author (Request Subsumed) : 0.252 *Publisher to Reader (Only Distance + WordNet) : 0.03625	0.28825
Service 4	*Foreign-Author to Reader (Only Distance) : 0.00018 *Publisher to Ordinary-Publisher (Service Subsumed) : 0.108	0.10818

Table 6.17. Final service rank for test case 2

Service Name	Service Score
Service 1	0.09306
Service 2	1.190256
Service 3	0.3745
Service 4	0.16689

As shown in the above table, we have the final ranking of services in the order: Service 2, Service 3, Service 4 and Service 1. This ranking aligns with the input and output parameter rankings.

As we compare the input parameter matching with a conventional matchmaker, Table 6.18 clearly shows that Service 1 is now left out as we do not have property-level matching anymore. This way we lose the connection between *Foreign-Magazine* and *Newspaper* concepts in Service 1 inputs. This is also the case in Service 4.

Table 6.18. Conventional matchmaker - Input parameter matching for test case 2

Service Name	Parameter Pairing	Service Score
Service 1	No match	0
Service 2	*Foreign-Magazine to Foreign-Magazine (Exact) : 0.9	0.9
Service 3	*Foreign-Magazine to Magazine (Request Subsumed) : 0.54	0.54
Service 4	No match	0

Table 6.19 shows that we have no differentiation for Service 1 and Service 4 anymore. This is because similarity distance annotations are not considered and Service 4 loses its advantage and importance in matchmaking. Besides, we do not have a fine-grained matchmaking as the previous cases.

Table 6.19. Conventional matchmaker - Output parameter matching for test case 2

Service Name	Parameter Pairing	Service Score
Service 1	*Publisher to Ordinary-Publisher (Service Subsumed) : 0.54	0.54
Service 2	*Foreign-Author to Author (Request Subsumed) : 0.36 *Publisher to Publisher (Exact) : 0.9	1.26
Service 3	*Foreign-Author to Author (Request Subsumed) : 0.36	0.36
Service 4	*Publisher to Ordinary-Publisher (Service Subsumed) : 0.54	0.54

The final rank with a conventional one points out that we do not have a separation for Service 1 and Service 4. It also shows that we do not have a fine-grained differentiation.

Table 6.20. Conventional matchmaker - Final service rank for test case 2

Service Name	Service Score
Service 1	0.324
Service 2	1.116
Service 3	0.432
Service 4	0.324

The above test runs demonstrate that matchmaking in SAM provides fine-grained differentiation for services, taking into account context information with similarity distance annotations. Besides, SAM can capture semantic similarity observing common properties in concepts. The advantages of SAM as compared to a conventional matchmaker will be far greater, when we have more complex service sets with a great number of services having parameters referring to several ontologies.

6.4. Threats to Validity

SAM addresses several challenges in the Web service matchmaking process. In this section we describe these challenges, describing how SAM overcomes them.

Considering input or output parameter matching, number of parameters in the request or service interface may not be equal. As mentioned in section 5, SAM applies normalization to similarity score if the input parameter count of a service is less than the request parameters.

The above case indeed raises an issue of imperfect matching. Although the service might satisfy the request with less input parameters, the interface of the service and request does not match perfectly. However, SAM does not make any classification of match type like perfect or imperfect. It just evaluates the amount of similarity score and provides a relative ranking between candidate services. But considering the parameter pairing SAM provides a match type with the similarity score. As mentioned before, the interpretation of scores and similarity distance information by the user is not considered in SAM. We assume that an automated system can provide the user with an interface to assign similarity distance values in a user-friendly way.

Bipartite graph matching in SAM does not differentiate between alternative maximum weight matching for a certain service and request pair. Indeed, a service and request pair might have more than one maximum weight matching, where some of them might be preferred to others. Such as a maximum weight matching that has equally distributed weights over parameter pairs can be preferred over a maximum matching, where the total weight is assigned only on a single parameter pair and remaining pairs have zero score. SAM graph matching module can be extended to consider this case and choose the best maximum weight matching in parameter pairing.

As SAM applies fine-grained similarity scoring with property-level matching some side-effects might occur. Although properties of two concepts might be similar, in some extreme cases the concepts might not have so much in common. This can result in introducing false positives in SAM matchmaking process. A validation step could be included after property-level matching to overcome this problem.

WordNet similarity score is considered as a second source of information. However, some parameter types defined in ontologies may not correspond to valid WordNet entries. In order to provide a fair matchmaking process WordNet scoring is enabled or disabled in SAM.

Matching and ranking involves the consideration of priorities. The similarity equations that are described by SAM make use of coefficients that represent priorities. Such as the output similarity score being prioritized over input similarity scores, as the outcome of a service is much more important for a request. Another example is the WordNet score having a less priority than the ontology based similarity distance score.

SAM is tested and evaluated on the book ontology provided with the OWL-S TC library. We presented two test scenarios with 9 services and two requests. Testing SAM with different ontologies and services may better demonstrate the value-added features of it. The coefficients in presented equations will converge to certain values as SAM is tested with different ontologies. However, the relative ratio of the coefficients will be very similar to current values. Current coefficient values are adjusted considering the book ontology and changing these values might affect the ranking of services. As stated, further testing will make SAM much more reliable.

6.5. Comparing SAM with Existing Matchmakers

As described in section 3, web services matchmaking has been an active research area. Several matchmakers have been introduced with prototype implementations. Each of these studies focus on certain aspects of matchmaking. In this section we will discuss the advantages and disadvantages of SAM in comparison to other matchmaker proposals.

In [55], authors Jian Wu and Zhaohui Wu, introduce the study “Similarity-based Web Service Matchmaking”. They present four similarity assessment methods as part of their matchmaker. As in SAM, they make use of WordNet but for a different purpose. SAM benefits from WordNet as a second source of information base for service parameter type similarity assessment. In their proposed matchmaker, WordNet is used to assess the similarity of service profile categories and parameter names instead of parameter types. Their approach deals with lexical similarity more than semantics. They consider the parameter type similarity through a data type similarity lookup table. SAM makes use of ontologies for parameter type similarity, which we believe is a better method considering semantics and standards. One advantage of their study is the consideration of QoS attribute similarity. However, SAM has the advantage of subsumption similarity assessment with a level down to properties of concepts and makes use of similarity distance information considering as the context in matchmaking.

In [73], one of the best known systems *Larks* is presented. The authors define an agent capability description language named *Larks* and deal with the problem of matchmaking. Their matchmaking algorithm makes use of this specialized language as opposed to SAM, where standard semantics technologies like OWL and OWL-S are employed. Besides, Larks has a discrete scale of similarity classification. On the other hand SAM assigns similarity scores and ranks the services. Larks ignore a service if one of the attributes is in a fail state. SAM just considers the total score a service retrieves, not eliminating and leaving the choice to the consumer. Besides, SAM makes use of WordNet, property-level matching and similarity distance information that Larks does not include.

In [74], Lei Li and Ian Harrocks introduce a service matchmaker prototype which works on DAML-S definitions. Their matchmaking algorithm uses a discrete scale of match as in Larks: Exact, plug-in, subsume, intersection and fail. One advantage over SAM is that their matchmaker provides service profile category match, where SAM assumes that the category match is performed and a candidate service set is presented. However, SAM has many advantages over Li and Harrocks's matchmaker such as property-level matching, WordNet scoring and use of similarity distance information to consider user preferences in matching.

In [54] authors introduce a matchmaking approach based on a custom capability description (OSDL) and query language (OSQL). However, different from other matchmakers they also consider property-level matching. They categorize the similarity of two concepts as equal, inherited (subsumption), property relation (one concept is a property of another) and mixed relation (a transitive relation where one concept is a property of another which is subsumed by a third concept). In their definition similarity of concept X to concept Y is not always equal to the reverse case. They argue that the similarity of subsuming concepts depends on the number of properties they possess. In SAM the similarity relation is symmetric. Besides, SAM does not consider concepts being properties of each other in matchmaking. We suspect that the property of a concept should contain similarities to the concept itself. SAM considers concepts having similar properties. Both SAM and their matchmaker focus on interface similarity but SAM introduces value-added approaches like similarity distance and WordNet scoring in addition.

Considering the above comparisons we can conclude that SAM contributes value-added approaches in matchmaking. Similarity distance information is used in a standard context improving MS-Matchmaker [24]. Property-level matching and WordNet scoring are all combined in a service ranking matchmaker. The following table summarizes this comparison.

Table 6.21. SAM in comparison to other matchmakers

Feature / Matchmaker	**SAM**	**MS-Matchmaker**	**Larks**	**Li & Harrocks**	**Wu & Wu**	**OSDL**
Subsumption	+	+	+	+	+	+
Service Interface Matching	+	+	+	+	+	+
Constraints	-	-	+	-	-	-
Property Level Matching	+	-	-	-	-	+
Similarity Distance	+	+	-	-	-	-
WordNet	+	-	-	-	+ (lexical)	-
Nonfunctional Matching	-	-	-	-	+	+
Service Category Matching	-	+	+	+	+	-
OWL-S	+	-	-	-	-	-

7. Conclusions and Future Work

We proposed a novel advanced matchmaker architecture, which introduces new value-added approaches like semantic distance based similarity assessment, property level assessment and WordNet similarity scoring. Instead of classifying candidate Web services in a discrete scale, our matchmaking agent applies a scoring scheme to rank candidate Web services according to their relevancy to the request.

The ranking property enables to include some of the relevant Web services in the final result set whereas they would have been discarded in a discrete scale classification. Additionally, our proposed matchmaking agent improves subsumption- based matchmaking by utilizing OWL constructs efficiently and by considering down to a level of concept properties in the process. An improvement at this point can be to consider similarity between properties in addition to similarity of property range objects.

We also introduced semantic distance annotation in ontology to represent relevancy of concepts to the user in a numerical way. Semantic distance annotations improve the relevancy of returned Web service set as they actually represent user's view of ontology. WordNet similarity measurement is also presented as a value-added feature, which acts as a secondary source of information, strengthening the power of reasoning.

Considering the value-added approaches that SAM introduces in matchmaking, we can conclude that in a service discovery architecture SAM will improve the precision and accuracy of the discovery process. As a result businesses can find partners in a distributed environment easily and with alternatives considered. This will lead to better collaboration among partners and improvement of business processes.

We think that preconditions and results of a service can also be considered for a complete matchmaking process. At that point, use of SWRL (Semantic Web Rule Language) in both service advertisements and request description will enhance the capabilities of our matchmaking agent. Current architecture of SAM supports such a rule based extension.

Non-functional attributes of Web services can also be taken into account in matchmaking process. QoS attributes of a Web service can be of great concern to a consumer. So, extending our matchmaking algorithm to include QoS attributes can be another improvement.

SAM currently presents a relative ordering of services as the output with similarity scores compared to a certain request. In order to eliminate some services with ignorable similarity scores a threshold value can be introduced. We leave this to the client agent that makes use of SAM. However, such a threshold can also be implemented in SAM as a cut-off score in order to classify only a subset of candidate service set as relevant.

Another improvement will be to add context aware decision-making capabilities, enabling our matchmaking agent to reason based on user profiles, preferences, past actions etc. The architecture that we have presented can be considered as a basis for the development of context-aware agent.

REFERENCES

1. Wang, H. and L. Zengzhi, L. Fan, An Unabridged Method Concerning Capability Matchmaking of Web Services, In Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, 2006.

2. Klusch, M., B. Fries, M. Khalid, and K. Sycara, OWLS-MX: Hybrid Semantic Web Service Retrieval, In 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web, AAAI Press, Arlington VA, 2005.

3. Keller, U., R. Lara, A. Polleres, I. Toma, M. Kifer and D. Fensel, WSMO Web Service Discovery, http://www.wsmo.org/2004/d5/d5.1/v0.1/20041112/.

4. El-Ghalayini, H., M. Odeh, R. McClatchey, and T. Solomonides, Reverse Engineering Ontology to Conceptual Data Models, In Proceeding (454) Databases and Applications, 2005.

5. Universal Discovery Description and Integration Protocol, http://www.uddi. org, 2006.

6. Semantic Web, W3C, http://www.w3.org/2001/sw/, 2006.

7. W3C, World Wide Web Consortium, http://www.w3.org/, 2006.

8. Khalid, M. A., Fries B. and Kapahnke P., OWL-S Service Retrieval Test Collection Version 2.1, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH Saarbrücken, Germany, 2006.

9. Saip, H. A. B. and C. L. Lucchesi, Matching Algorithms for Bipartite Graphs, Relatorio Tecnico DCC-03/93.

10. Motta, E., J. B. Domingue, L. S. Cabral and M. Gaspari, IRS-II: A Framework and Infrastructure for Semantic Web Services, Proceedings of the 2nd International Semantic Web Conference (ISWC2003), Vol. 2870 of LNAI., Springer, pp.306-318, Florida, USA, 2003.

11. OWL-S Submission, http://www.w3.org/Submission/OWL-S, 2004.

12. Fensel, D. and C. Bussler, The Web Service Modeling Framework: WSMF, Electronic Commerce: Research and Applications, Vol. 1, No. 2, pp. 113-137, 2002.

13. Klein, M., B. Konig-Ries and M. Muussig, What is needed for semantic service descriptions? A proposal for suitable language constructs, Proceedings of Inernational Journal of Web and Grid Services, Vol. 1, No. 3/4, pp. 328-364, 2005.

14. Voorhees, E., Using WordNet for Text Retrieval, C. Fellbaum(Editor), WordNet: An Electronic Lexical Database, pp. 285-303, The MIT Press, Cambridge,1998.

15. Ginsberg, A., A Unified Approach to Automatic Indexing and Information Retrieval, IEEE Expert , Vol. 8, No. 5, pp 46-56, 1993.

16. Lee, J., M. Kim and Y. Lee, Information Retrieval Based on Conceptual Distance in IS-A Hierarchies, Journal of Documentation, Vol. 49, No. 2, pp. 188-207, 1993.

17. Agirre, E. and G. Rigau, Word Sense Disambiguation Using Conceptual Density, Proceedings of the 16th Conference on ComputationalLlinguistics, Vol.1, pp. 16-22, 1996.

18. Hovy, E., Combining and Standardizing Large-scale, Practical Ontologies for Machine Translation and Other Uses, Proceedings of the 1st International Conference on Language Resources and Evaluation (LREC), Granada, Spain, 1998.

19. Wang, Y. and E. Stroulia, Semantic Structure Matching for Assessing Web-Service Similarity, Proceedings of the 1st International Conference on Service Oriented Computing, Trento, Italy, 2003.

20. Guo, R., D. Chen and J. Le, Matching Semantic Web Services accross Heterogeneous Ontologies, Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT'05), 2005.

21. Paolucci, M., T. Kawamura, T. Payne, and K. Sycara, Semantic matching of Web services capabilities. In Horrocks, I. And Hendler, J.eds.Proc. of the 1st International Semantic Web Conference(ISWC), pages333-347. Springer, 2002.

22. Shen, X., X. Jin, R. Bie and Y. Sun, MSC: A Semantic Ranking for Hitting Results of Matchmaking of Services, Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06).

23. Osman T., D. Thakker, D. Al-Dabass, Semantic-Driven Matchmaking of Web Services Using Case-Based Reasoning, IEEE International Conference on Web Services, 2006.

24. Şenvar, M. and A. Bener, Matchmaking of Semantic Web Services Using Semantic- Distance Information, Lecture Notes in Computer Science by Springer Verlag, ADVIS 2006, October, 18-20, İzmir, Turkey.

25. Srinivasan, N., M. Paolucci and K. Sycara, Semantic Web Service Discovery in the OWL-S IDE, Proceedings of the 39th Hawaii International Conference on System Sciences, 2006.

26. Wang, H. and Z. Li, A Semantic Matchmaking Method of Web Services Based On SHOIN+(D), Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06).

27. Paolucci, M., T. Kawamura, T. R. Payne and K. Sycara, Semantic Matching of Web Services Capabilities, International Semantic Web Services Conference, 2002.

28. Bunke, H., Graph Matching: Theoretical Foundations, Algorithms and Applications, in Proc. Vision Interface 2000, Montreal, 2000, 82 – 88.

29. Jena – A Semantic Web Framework for Java, http://jena.sourceforge.net/.

30. McBride, B., Jena: A Semantic Web Toolkit, IEEE Internet Computing, 2002.

31. CMU OWL-S API, http://www.daml.ri.cmu.edu/owlsapi.

32. OWL-S API, http://www.mindswap.org/2004/owl-s/api/doc/.

33. Resource Description Framework (RDF), http://www.w3.org/RDF/, 2004.

34. Castillo, J. G., D. Trastour and C. Bartolini, Description Logics for Matchmaking of Services, Trusted E-Services Laboratory HP Laboratories Bristol, 2001.

35. Daconta, M. C., L. J. Obrst and K. T. Smith, The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management, John Wiley & Sons, 2003.

36. Semantic Web Vision, http://www.hpl.hp.com/semweb/sw-vision.htm.

37. Shadbolt, N., and W. Hall and T. B. Lee, The Semantic Web Revisited, IEEE Intelligent Systems, 2006.

38. Decker, S., P. Mitra and S. Menlik, Framework for the Semantic Web: An RDF Tutorial, IEEE Internet Computing, 2000.

39. OWL Web Ontology Language Overview, http://www.w3.org/TR/owl-features/, 2004.

40. OWL Web Ontology Language Guide, http://www.w3.org/TR/owl-guide/, 2004.

41. OWL-S 1.1 Release, http://www.daml.org/services/owl-s/1.1/.

42. OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.1/overview/.

43. Balzer, S., T. Liebig and M. Wagner, Pitfalls of OWL-S – A Practical Semantic Web Use Case, ICSOC, 2004.

44. Jørstad, I., S. Dustdar and D. V. Thanh, A Service Oriented Architecture Framework for Collaborative Services, Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, 2005.

45. Hashimi, S., Service-Oriented Architecture Explained, http://www.ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html, 2003.

46. Pasley, J., How BPEL and SOA are Changing Web Services Development, IEEE Internet Computing, 2005.

47. Service Oriented Architecture, http://msdn2.microsoft.com/en-us/architecture/aa948857.aspx.

48. Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl, 2001.

49. SOAP Version 1.2 Part 0: Primer, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, 2003.

50. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), http://www.w3.org/TR/soap12-part1/, 2007.

51. Extensible Markup Language (XML), http://www.w3.org/XML/.

52. Chester, T. M., Cross-Platform Integration with XML and SOAP, IT PRO, 2001.

53. Introduction to UDDI: Important Features and Functional Concepts, OASIS, http://uddi.xml.org/, 2004.

54. Kuang, L., J. Wu, S. Deng, Y. Li, W. Shi and Z. Wu, Exploring Semantic Technologies in Service Matchmaking, Proceedings of the Third European Conference on Web Services, 2005.

55. Wu, J. and Z. Wu, Similarity-based Web Service Matchmaking, Proceedings of the 2005 IEEE International Conference on Services Computing, 2005.

56. Sirin, E., B. Parsia and J. Hendler, Composition-driven Filtering and Selection of Semantic Web Services, In AAAI Spring Symposium on Semantic Web Services, March 2004.

57. Akkiraju, R., B. Srivastava, A. Ivan, R. Goodwin and T. S. Mahmood, Semantic Matching to Achieve Web Service Discovery and Composition, Proceedings of the 8th IEEE International Conference on E-Commerce Technology and the 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2006.

58. Defining N-ary Relations on the Semantic Web, http://www.w3.org/TR/swbp-n-aryRelations/, 2006.

59. OASIS-UDDI, http://www.uddi.org/.

60. Business Process Execution Language for Web Services version 1.1, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2007.

61. Schmitz, D., G. Lakemeyer, G. Gans and M. Jarke, Using BPEL Process Descriptions for Building up Strategic Models for Inter-Organizational Networks, In International Workshop on Modeling Inter-Organizational Systems (MIOS), Springer, LNCS, Larnaca, Cyprus, October 2004.

62. Miller, G. A., WordNet: A Lexical Database for English, Communicating of the ACM, 1995.

63. Petersen, T., S. Patwardhan and J. Michelizzi, WordNet::Similarity - Measuring the Relatedness of Concepts, In Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04), 2004.

64. Liu, P. Y., T. J. Zhao and X. F. Yu, Application-Oriented Comparison and Evaluation of Six Semantic Similarity Measures Based on WordNet, Proceedings of the Fifth International Conference on Machine Learning and Cybernetics, Dalian, 2006.

65. Yao, Y., S. Su and F. Yang, Service Matching based on Semantic Descriptions, Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, 2006.

66. Guo1, R., J. Le and X. L. Xia, Capability Matching of Web Services Based on OWL-S, Proceedings of the 16th International Workshop on Database and Expert Systems Applications, 2005.

67. Colucci, S., T. D. Noia, E. D. Sciascio, F. M. Donini and M. Mongiello, Description Logics Approach to Semantic Matching of Web Services, 25th Int. Conf. Information Technology Interfaces, 2003.

68. Lim, J. E., O. H. Choi, H. S. Na and D. K. Baik, A Methodology for Semantic Similarity Measurement among Metadata based Information System, Proceedings of the Fourth International Conference on Software Engineering Research, 2006.

69. Marcus, A. and J. I. Maletic, Identification of High-Level Concept Clones in Source Code, In Proceedings Automated Software Engineering (ASE'01), San Diego, CA, November 2629 2001, pp. 107-114.

70. OASIS, www.oasis-open.org.

71. Ma, K. J., Web Services: What's Real and What's Not?, IEEE IT PRO, 2005.

72. Munkres' Assignment Algorithm, http://www.public.iastate.edu/~ddoty/HungarianAlgorithm.html.

73. Sycara, K., M. Klusch, S. Widoff, and J. Lu, Dynamic Service Matchmaking Among Agents in Open Information Environments, SIGMOD Record, Vol. 28, No.1, March 1999.

74. Li, L. and I. Harrocks, A Software Framework For Matchmaking Based on Semantic Web Technology, In Proceedings of the Twelfth International World Wide Web Conference (WWW 2003), 2003.

75. Ilhan, E. S., G. B. Akkus and A. B. Bener, SAM: Semantic Advanced Matchmaker,The Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'07), Boston, USA, July 2007.

76. Ilhan, E. S., G. B. Akkus and A. B. Bener, Improved Service Ranking and Scoring: Semantic Advanced Matchmaker (SAM), 2nd International Working Conference on Evaluation of Novel Approaches to Software Engineering, Barcelona, Spain, July 2007.