PRECONDITION AND EFFECT MATCHING USING SWRL

by

Volkan Özadalı

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
2008

ACKNOWLEDGEMENTS

Foremost, I would like to thank and express my gratitude to my supervisor, Assistant Prof. Ayşe Başar Bener, who shared with me a lot of her expertise and research insight. And I am deeply grateful to her for the trust and support that she gave me in order to complete my thesis. I would also like to thank Erdem Savaş İlhan for his support while implementing my thesis.

I am also so grateful to my manager and colleagues for their understanding throughout this study.

Finally, I wish to express my love and gratitude to my family and my love Tuğba Şimşek.

ABSTRACT

PRECONDITION AND EFFECT MATCHING USING SWRL

Service oriented architectures provide more effective and dynamic applications. Using Semantic Web services in service oriented architectures improves interoperability and scalability. A very important aspect of using Semantic Web services is the matchmaking process. Semantic matchmaking is used during discovery and composition of Semantic Web services to find valuable service candidates. Among these candidates, best ones are chosen to build up the composition, or for substitution in the case of an execution failure.

In this research we enhance Semantic Advanced Matchmaker (SAM) which is based on input and output matching. Our new matchmaking agent supports precondition and effect expressions written in SWRL during matchmaking in addition to input and output annotations. We present a novel approach for assigning matchmaking scores to condition expressions in OWL-S documents written in SWRL language. Proposed matchmaking method utilizes subsumption-based similarity, property-level similarity, similarity distance annotations and WordNet-based similarity. The algorithm uses bipartite graphs in order to find matching parameters of requests and advertisements and then ranks the advertisements according to their semantic similarity to the request.

Using classical information retrieval evaluation techniques we show that our proposed agent, which is capable of precondition and effect matching, has significantly higher precision performance with respect to SAM on input and output matching.

ÖZET

SWRL TANIMLI ÖN ŞART VE ETKİ EŞLEYİCİ

Hizmet tabanlı mimari daha etkili ve dinamik uygulamalara zemin sağlamaktadır. Hizmet tabanlı mimari bünyesinde anlamsal Web hizmetlerinin kullanımı, birlikte işlerliği ve esnekliği arttırmaktadır. Anlamsal Web hizmetlerinin kullanımıyla ilgili önemli noktalardan birisi de kullanılan hizmet eşleme yöntemidir. Anlamsal eşleme, önemli hizmet adaylarının bulunabilmesi için, anlamsal Web hizmetlerinin eşleştirilmesi ve kompozisyonu sırasında kullanılmaktadır. Bu adaylar arasından en iyileri seçilerek kompozisyon oluşturulabilir ya da bir çalışma hatası anında ilgili hizmetin yerine seçilen hizmet kullanılabilir.

Bu araştırmada, önceden yapılan girdi-çıktı bazlı servis eşleme işlemsel süreci daha gelişmiş bir sürece dönüştürülmektedir. Yeni eşleştirme süreci, girdi-çıktı tanımlarına ek olaraktan SWRL diliyle yazılmış ön şart ve etki ifadelerini de desteklemektedir. Bu araştırmada, SWRL diliyle yazılmış OWL-S koşul ifadelerine eşleştirme puanı belirlemek için yeni bir yöntem sunulmaktadır. Önerilen eşleme metodu, kapsama tabanlı benzerlik, özellik tabanlı benzerlik, benzerlik uzaklığı notları ve WordNet tabanlı benzerlik yöntemlerini kullanmaktadır. Sunulan işlemsel süreç, isteğin ve sunulan hizmetlerin eşleşen özelliklerini bulup, eşleşen servisleri isteğe olan anlamsal benzerliklerine göre sıralamak için iki kümeli grafik bazlı yöntemlerden faydalanmaktadır.

Geleneksel bilgi sağlayıcı değerlendirme yöntemleri kullanılarak geliştirilen işlem sürecinin, ön şart ve etki ifadelerini destekleyen, önceki girdi-çıktı bazlı işlem sürecinden daha iyi bir doğruluk değerinin olduğu gösterilmektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABox	Assertional Box
API	Application Programming Interface
DAML	DARPA Agent Markup Language
DL	Description Logic
HTTP	Hypertext Transport Protocol
IO	Input, Output
IOPE	Input, Output, Precondition, Effect
IR	Information Retrieval
IT	Information Technology
KIF	Knowledge Interchange Format
MAP	Mean Average Precision
OIL	Ontology Interchange Language
OWL	Web Ontology Language
OWL-S	OWL based Web service Ontology
OWL-S TC	OWL-S Service Retrieval Test Collection
PE	Precondition, Effect
QOS	Quality of Service
PDDL	Planning Domain Definition Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RuleML	Rule Markup Language
SAM	Semantic Advanced Matchmaker
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SWRL	Semantic Web Rule Language
SWWS	Semantic Web Enabled Web Services
Tbox	Terminological Box
UDDI	Universal Description, Discovery, Integration
URI	Universal Resource Identifier
URL	Uniform Resource Locator

WSDL	Web Services Description Language
WSDL-S	Web Service Semantics
WSMO	Web Service Modeling Ontology
WWW	Word Wide Web
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1. INTRODUCTION

1.1. Motivation

Service Oriented Architecture (SOA) is an architectural style that guides all aspects of creating and using business processes, packaged as services, throughout their lifecycle, as well as defining and provisioning the IT infrastructure that allows different applications to exchange data and participate in business processes loosely coupled from the operating systems and programming languages underlying those applications [1]. A service can be defined as a unit of work done by a service provider to accomplish desired results for a service consumer. Service oriented architectures are becoming popular since they provide more effective and dynamic applications. One of the most widely used and accepted representative of the SOA paradigm is Web services. A Web service is a is a self-describing, self-contained, modular unit of application logic that provides some piece of business functionality to other applications over the Web, is described via WSDL [2] and is capable of being accessed via standard network protocols such as SOAP [3, 54] over HTTP [4].

A transaction with a service requires at least two parts: the service requester seeking a service to complete its work, and the service provider providing a service sought by the user [78]. In order to fulfill service requests, a match needs to be determined between the requested service and one of the available services (advertisements). Matching can be defined as the process to determine a relation between the user request and the advertisements [34]. This process is the main area of investigation of this work. In case no perfect match is found between the requested and the advertised services, composite services may be formed using various available services, or partial matches can be used. The output of the matching procedure can be one perfect matching service, or many partially matching services ordered according to some ranking criteria [34].

The Semantic Web is an evolving extension of the WWW (World Wide Web) in which Web content can be expressed not only in natural language, but also in a format that can be read and used by software agents, thus permitting them to find, share and integrate information more easily [5]. Using Semantic Web services in service oriented architectures improves interoperability and scalability. It is the common matching practice adopted currently. OWL [6] is a XML [60] based language having the capability of specifying semantics. OWL specifies semantics by supporting Description Logics (DL) [7]. In terms of providing meaning to the content presented in Web services, a standard has emerged that describes the semantics for Web Services. This standard was developed by the DAML Coalition and is now called OWL-S [8]. OWL-S (Semantic Markup for Web Services) [1] is an OWL based attempt for standardization of Web service descriptions to establish a framework within which these descriptions are made and shared. OWL-S provides service profile class which includes IOPE (input, output, precondition, effect) capabilities of the services. During the matchmaking process, requests and advertisements (service offers) described as OWL-S documents are matched according to IOPE capabilities and the best advertisement is selected or a list of matching results is generated to make the choice manually. The difficulty in the matchmaking process occurs when there is no exact match for the request. Partial matches must be evaluated in these situations.

The SWRL (Semantic Web Rule Language) [9] is a rule language that combines OWL with the rule markup language [10] providing a rule language compatible with OWL. SWRL expressions may be used in OWL-S preconditions, and in effects expressions. Using SWRL makes the OWL-S ontologies more powerful since it uses the expressive power of rules and powerful reasoning options [83].

IO matching and PE matching show differences due to their different natures. IO attributes describe syntactically which inputs are required by a service to function, and which outputs are returned; whereas PE rules describe under which conditions the service can be executed and the changes in the world after the execution of the service. Relatively less works exist on PE concerning matching. In this paper, we extend the previous matchmaking agent, Semantic Advanced Matchmaker (SAM), to support PE matchmaking [11]. We propose a capability matchmaking algorithm specialized for matchmaking needs of PE rules. The main contribution of our algorithm is that it provides ordered ranking

based solution for peculiar needs of PE matchmaking on OWL-S documents written in SWRL. Proposed matchmaking method utilizes subsumption-based similarity, property-level similarity, similarity distance annotations and WordNet-based similarity. The algorithm uses bipartite graphs in order to find matching parameters of requests and advertisements and then ranks the advertisements according to their semantic similarity to the request.

1.2. Outline

This thesis is outlined as follows. In Section 2, we first provide an overview of the background for this work, including the vision of the Semantic Web and discuss the standards and technologies used to bring this vision into being. The major building blocks are Description Logics, RDF(S), OWL, OWL-S, and SWRL which form the foundation of the idea presented in this thesis. We will also give a brief introduction to the tools and methods used, namely: bipartite graph matching, Jena, OWL-S API, WordNet, WordNet::Similarity, IR evaluation methods, and Mann-Whitney significance test.

In Section 3, we briefly review the previous researches related to this work. 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 is devoted to our approach to service matchmaking. We give details of our proposed approach and introduce the techniques and methods we apply in formalism. In Section 6, we describe the test environment that is used to evaluate and compare the proposed matchmaker with the previously done matchmaker. We present and discuss the statistics collected from the experiments in this section and also list the threats to validity of the proposed matchmaker.

Finally, in Section 7 we conclude the work in this thesis and we give an outlook on future research directions.

2. SEMANTIC WEB SERVICES

The initial Web service technology stack around SOAP, WSDL, and UDDI has significant deficiencies, especially with respect to automated discovery, composition, and usage of Web services. In order to overcome this, semantically enabled technologies are developed on basis of exhaustive frameworks and ontologies as the underlying data model. In this section we will give an overview of these technologies used in the emerging concept of Semantic Web services.

2.1. Semantic Web

The Semantic Web is a machine-processable web of smart data. Smart data is application-independent, composable, classified, and part of a larger information ecosystem or ontology. Figure 2.1 shows the progression of data along a continuum of increasing intelligence. The four stages in the diagram progress from data with minimal intelligence to data embodied with enough semantic information to allow us to make inferences about it [14].

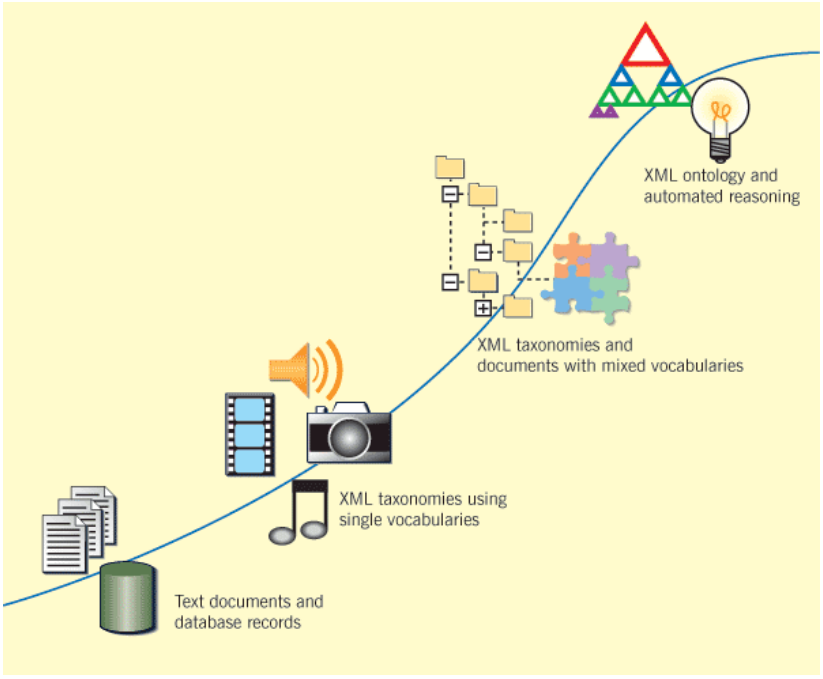

Figure 2.1. The smart data continuum [14]

- Text and databases (pre-XML): Most data are proprietary to an application. Thus, the smarts are in the application and not in the data.

- XML documents for a single domain: Data achieves application independence within a specific domain, with sufficient smarts to move between applications in a single domain. Examples of this are the XML standards in the health care industry, insurance industry, or real estate industry.

- Taxonomies and documents with mixed vocabularies: Data can be composed from multiple domains and classified accurately in a hierarchical taxonomy. In fact, the classification can be used for discovery of data. Simple relationships between categories in the taxonomy can be used to relate and thus combine data. Data is, therefore, now smart enough to be discovered easily and combined sensibly with other data.

- Ontologies and rules: New data can be inferred from existing data by following logical rules. In essence, data are now smart enough to be described with concrete relationships and sophisticated formalisms, where logical calculations can be made on this *semantic algebra*, which allows the combination and recombination of data at a more atomic level and very fine-grained analysis of data. In this stage, data no longer exists as a blob but as a part of a sophisticated microcosm. An example of this data sophistication is the automatic translation of a document in one domain to the equivalent (or as close as possible) document in another domain.

Description Logics and OWL are the foundation for providing semantics to Web resources. In order to provide semantics to a resource, some additional knowledge needs to be presented along with the resource. This creates the need for a knowledge representation formalism to be used. In the context of the Semantic Web [57], the knowledge representation formalism that is used is known as Description Logics and the language is the Web Ontology Language (OWL).

2.1.1. Description Logics

Description logics (DLs) are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way [23]. Description logics can be characterized by the following three properties:

- The basic syntactic building blocks are atomic concepts, atomic roles and individuals.

- The expressive power of the language is restricted by using a rather small set of constructors for building complex concepts and roles from existing concepts and roles.

- Implicit knowledge about concepts and individuals can automatically be found with reasoning techniques.

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.

2.1.2. RDF

The Resource Description Framework (RDF) provides basic fact-stating facilities [24, 25]. With RDF, any resource (identified through an URI) can be described. Similarly to natural languages, these descriptions follow the scheme subject, predicate, object. This scheme is called a triple.

On top of RDF, there exists an extension called RDF Schema. RDF extended by RDF Schema (RDFS) can be seen as a vocabulary description language providing class and property-structuring facilities [61]. Hence, it is possible to define classes, sub-classes and

properties of classes. This allows a more flexible way of structuring a vocabulary compared to simple fact-stating of RDF without RDFS.

RDF(S) alone has not the complete expressiveness of a typical ontology representation language. A number of logical constraints cannot be defined using RDF(S). This results in the inability of expressing the following cases (amongst others) [15]:

- Logical combinations of classes (intersection, union, complement).
- Advanced attributes of properties (transitive, symmetric, functional, inverse).
- Restrictions on local properties (e.g. one value must come from a particular class or at most 11 values are allowed).
- Equivalence and disjointness of classes.

2.1.3. OWL

The Web Ontology Language (OWL) is an ontology representation language standardized by the W3C. OWL provides a XML based realization of description logics [6]. Ontology representation languages have evolved over the years. Originally, two major projects were involved in the development of two separate ontology representation languages, which later were incorporated into OWL. These projects were those of the Ontology Interchange Language (OIL) and the DARPA Agent Markup Language (DAML). Both projects were merged into the DAML+OIL project. The resulting ontology representation language was submitted to the World Wide Web Consortium and was adopted by the WebOnt working group as the basis for their development of the Web Ontology Language [62]. OWL is conceptionally based on top of RDFS. All major classes of OWL are derived from corresponding RDFS base classes.

Since the universal expressivity has a critical impact on the practical computability, three differently powerful variations of OWL are available [16]. These variations are:

OWL Full, OWL DL (Description Logic), and OWL Lite. OWL Full includes all language primitives and, thus, poses the most powerful OWL variation. Unfortunately, this high level of expressiveness leads to computational undecidability. Advantages about OWL Full is, however, that it is the only variation fully upward compatible with RDF i,.e. any correct RDF document is, at the same time, a correct OWL Full document. OWL DL on the other hand is not fully compatible with RDF(S) anymore. The language is partially restrained, enabling efficient reasoning support. The least powerful variation is OWL Lite. OWL Lite is a real subset of OWL DL. It is supposed to be easier to learn and was also intended to push the development of tools [15].

2.2. Semantic Web Services

Web services are software services identified by a URI that are described, discovered, and accessed using Web protocols [63]. Web services consume and produce XML. As Web services proliferate, they become similar to Web pages in that they are more difficult to discover. The Semantic Web vision, as applied to Web services, aims at automating the discovery, invocation, composition, and monitoring of Web services by providing machine interpretable descriptions of services [64]. Besides the pure syntactic handling of Web services the combination of Semantic Web technologies and Web service technologies lead to a new and very powerful instrument: Semantic Web (enabled) Web services (SWWS). Figure 2.2 illustrates the combination of technologies [22].

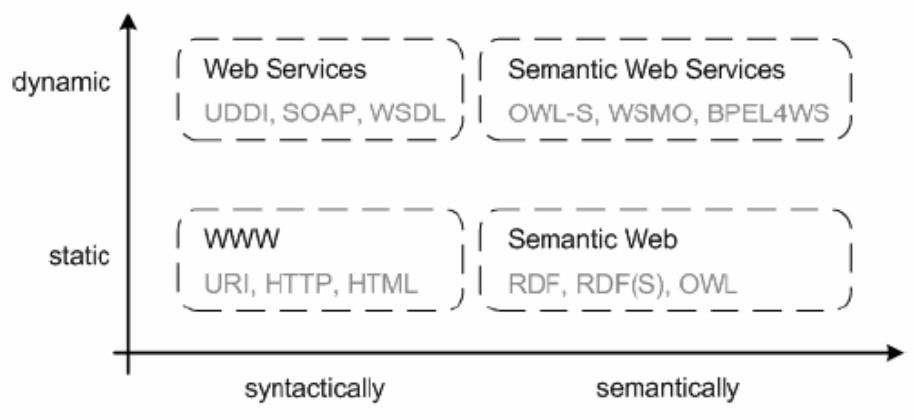

Figure 2.2. Semantic Web Services [22]

2.2.1. OWL-S

Any given service has a number of functional and non-functional properties. Different technologies exist that enable semantic annotation of Web services. Currently, none of these technologies has been declared as standard by W3C so far [65]. However, we will describe and use OWL-S, whereas competing technologies are WSDL-S (Web Service Semantics) [16] and WSMO (Web Service Modeling Ontology) [17].

An ontology representation language such as OWL can also be used in order to define an upper ontology capturing different aspects that are orthogonal to specific domain aspects [15]. OWL-S is an upper ontology for services. Therefore, it provides a set of concepts necessary for the semantic description of Web services. According to OWL-S, a service is described by the following aspects: Service Profile, Service Model, and Service Grounding.

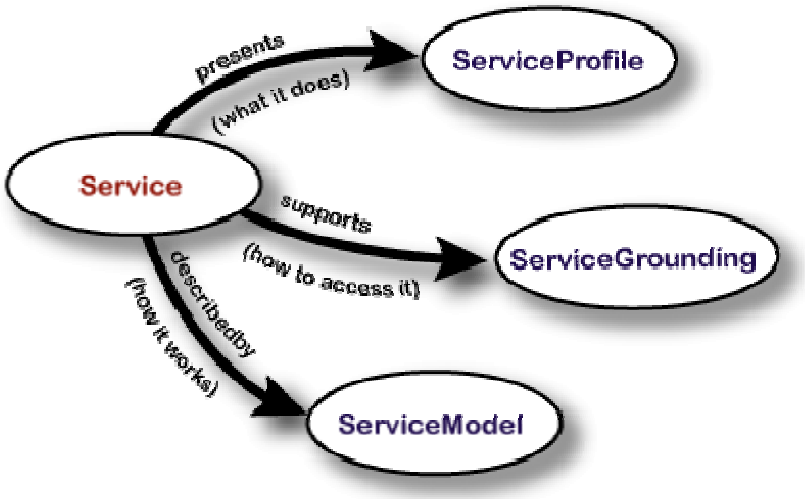

Figure 2.3. OWL-S ontologies [8]

The Service Profile is intended to be used as an advertisement for a service in a service repository. It provides information about the service, which is rather general [8]. Accordingly, this information is only used for selection of a service and not for detailed planning of interaction with it. Information captured by the Service Profile includes contact information of the service provider, a categorization of the service, a set of non-functional properties such as QoS constraints, and, finally, a functional description of the service. The

functional description specifies inputs, outputs, preconditions, and effects, which often are collectively labeled as IOPEs.

After selection of a service based on the Service Profile, the Service Model is consulted for information on the interaction with the service. An available interaction scenario with the service is called a process. The provided process description is used by the requestor to coordinate interaction with the service. There are three different types of processes (interaction scenarios) available: atomic process, composite process, simple process. Atomic processes can be directly invoked and executed in a single step. A composite process consists of a number of logical steps, which again represent any kind of process. A simple process should be chosen in case it is desired to provide a different abstract view on an atomic process or to provide a simplified representation of a composite process. Similarly to the Service Profile, the process description also contains information about IOPEs. However, these might be defined more fine granular than those in the Service Profile. In any case, consistency between the two descriptions has to be maintained either manually or by a supporting tool. Preconditions and effects have to be defined using logical formulas (using e.g. KIF [18], PDDL [19], SWRL [20]).

In OWL-S a Service Grounding defines how the abstract semantic description of a service is to be interpreted in order to technically execute the actual service. This is accomplished by defining how to format inputs and outputs as messages and how to exchange these messages. Grounding OWL-S on WSDL is very common. OWL-S suggests the complementary use of OWL-S and WSDL. That is, OWL-S aspects can be mapped onto aspects of WSDL. An OWL-S atomic process can be mapped onto a WSDL operation for instance. OWL-S in- and outputs can, furthermore, be mapped onto (parts of) WSDL in-/output messages of an operation [84].

2.2.2. SWRL

SWRL (Semantic Web Rule Language) is based on a combination of the OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language [20]. SWRL includes a high-

level abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sublanguages of OWL. An extension of the OWL model-theoretic semantics is also given to provide a formal meaning for OWL ontologies including rules written in this abstract syntax. SWRL can be used in preconditions and effect (rule) descriptions of OWL-S documents [66].

The proposed rules are of the form of an implication between an antecedent (body) and consequent (head) [20]. The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Both the antecedent (body) and consequent (head) consist of zero or more atoms. An empty antecedent is treated as trivially true (i.e., satisfied by every interpretation), so the consequent must also be satisfied by every interpretation; an empty consequent is treated as trivially false (i.e., not satisfied by any interpretation), so the antecedent must also not be satisfied by any interpretation. Multiple atoms are treated as a conjunction. Note that rules with conjunctive consequents could easily be transformed (via the Lloyd-Topor transformations) into multiple rules each with an atomic consequent. Atoms in these rules can be of the form C(x), P(x,y), sameAs(x,y) or differentFrom(x,y), where C is an OWL description, P is an OWL property, and x, y are either variables, OWL individuals or OWL data values. An XML syntax is also given for these rules based on RuleML and the OWL XML presentation syntax [67]. Furthermore, an RDF concrete syntax based on the OWL RDF/XML exchange syntax is presented. In OWL-S descriptions we use RDF concrete syntax [68].

As a rule language for the Semantic Web, SWRL uses URIs to identify things, making it essentially compatible with RDF and OWL. In RDF, an organization can express that a particular person in an employee and is also granted access to all internal documents. In SWRL, one can express the rule that all employees are granted access to internal documents. Given this rule and the fact that someone is an employee, a SWRL reasoner can conclude that the person is granted access. SWRL is unique in being an extension of OWL DL, so that users of OWL DL can add rules to their ontologies and maintain clear semantics. Some rule systems offer meta-processing (rules about rules), and with the addition of OWL comes the possibility for new confusion in rules about OWL axioms and OWL axioms about rules; the design of SWRL 0.6 carefully steers clear of these potentially-confusing areas. The development of languages like SWRL, with high

expressive power, also raises the question of the computational complexity of the implementation: as the submission authors correctly note, there will be the need to select suitable subsets of the language that can be implemented in efficient ways, balancing expressive power against execution speed and termination of the computation [21].

An example SWRL rule that is used to represent the precondition of a service is given in Table 2.1 from our test collection. In this rule *IndividualPropertyAtom* class of SWRL is used. It consists of a *propertyPredicate* and two arguments. Other possible classes for representing precondition expressions are *ClassAtom, DatavaluedPropertyAtom, SameIndividualAtom, DifferentIndividualsAtom,* and *BuiltinAtom*. Precondition in the example has the name *CustomerHasVisaCard* and checks whether a customer has a visa card. Arguments are of class *CustomerNum* and *VisaCard* and the predicate is identified with the *hasCard* property predicate. Definitions of the arguments and the property predicate must be specified or imported in the OWL-S document. A service may have more than one precondition, and each precondition may have more than one property predicate.

Table 2.1. SWRL precondition example

```
<profile:hasPrecondition rdf:resource=”#CustomerHasVisaCard”/>
...
<expr:SWRL-Condition rdf:ID=”CustomerHasVisaCard”>
        <rdfs:label>hasCard(CustomerNum, VisaCard)</rdfs:label>
        <expr:expressionLanguage rdf:resource=”&expr;#SWRL”/>
        <expr:expressionObject>
                <swrl:AtomList>
                        <rdf:first>
                                <swrl:IndividualPropertyAtom>
                                        <swrl:propertyPredicate rdf:resource=”#hasCard”/>
                                        <swrl:argument1 rdf:resource="#CustomerNum"/>
                                        <swrl:argument2 rdf:resource="#VisaCard"/>
                                </swrl:IndividualPropertyAtom>
                        </rdf:first>
                        <rdf:rest rdf:resource="&rdf;#nil"/>
                </swrl:AtomList>
        </expr:expressionObject>
</expr:SWRL-Condition>
```

Execution of a service can generate different outputs and domain changes (effects) in different conditions. Result specification in OWL-S allows definitions of conditional outputs and effects. An example result description is given in Table 2.2 from our test collection. This example is not complete, only the condition names are given in *inCondition* and *hasEffect* parts. This example states that if the card operation is successful then an acknowledgement message *CardAcceptMsg* is given as output and the world state is changed to record the information that the *VisaCard* of *CustomerNum* has been accepted. Changes in the world state can be utilized in Web service composition to determine the satisfiability of the preconditions of the other Web services that participates.

Table 2.2. SWRL result example

```
<profile:hasResult rdf:resource="#CardOperationResult"/>
...
<process:Result rdf:ID="CardOperationResult">
        <process:inCondition>
                <expr:SWRL-Condition>
                        CardOperationResult(CustomerNum,VisaCard)
                </expr:SWRL-Condition>
        </process:inCondition>
        <process:withOutput rdf:resource="#Ack">
                <valueType rdr:resource="#CardAcceptMsg">
        </process:withOutput>
        <process:hasEffect>
                <expr:SWRL-Condition>
                        CardAccepted(CustomerNum,VisaCard)
                </expr:SWRL-Condition>
        </process:hasEffect>
</process:Result>
```

2.3. Complementary Tools and Methods

In this section we describe the main complementary tools and methods that we utilized while implementing and evaluating our research. In each subsection we will give a brief overview of each topic and describe its connection to our research.

2.3.1. Bipartite Graph Matching

In graph theory, a bipartite graph is an undirected graph in which each of its vertices falls in one of two sets: the left set *L*, or the right set *R* [69]. There can only be edges between vertices of different set, and each edge may be assigned a weight. A matching is a subset of edges *M*, such that for all vertices at most one edge of *M* is an incident. A maximum cardinality match is a match that has the most number of edges among all matches. A maximum weight match is a matching such that the sum of the weights of the edges in the matching is maximized. The maximum cardinality maximum weight bipartite matching problem is about finding the maximum cardinality matching *M* that has the maximum weight sum. Figure 2.4 depicts a maximum weight matching example.

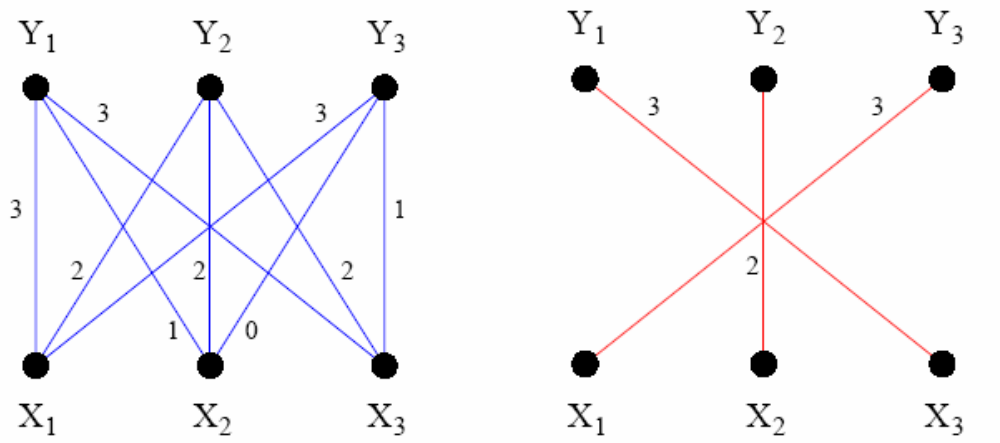

Figure 2.4. Maximum weight matching

Hungarian algorithm is invented and published by Harold Kuhn in 1955, works on weighted bipartite graphs to find a minimum cost maximum matching [26]. 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 *n* x *m* cost matrix, where each element of the matrix represents the cost associated with assigning a certain job to the specified resource [85].

In our research we use bipartite graph matching to figure out which parameter in one advertisement profile corresponds to a parameter in the request profile (separately for each IOPE). This process is called parameter pairing. We use the Hungarian algorithm for this process, since it is the best known strongly polynomial time bound algorithm for weighted

bipartite matching that it runs in time $O(|V| |E| + |V| \log|V|)$ where V represents the number of vertices and E represents the number of edges [26].

2.3.2. Jena

Jena is an open source Java API for Semantic Web applications [58, 59]. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract model. A model can be sourced with data from files, databases, URLs or a combination of these. The API has been defined in terms of interfaces so that application code can work with different implementations without change [70]. The package contains interfaces for representing models, resources, properties, literals, statements and all the other key concepts of RDF. Jena provides support for OWL (Web Ontology Language). The framework has various internal reasoners and also provides support for external reasoners. In addition the Pellet [27] reasoner (an open source Java OWL-DL reasoner) can be set up to work in Jena.

2.3.3. OWL-S API

OWL-S API provides a Java API for programmatic access to read, execute and write OWL-S service descriptions [28]. The API supports to read different versions of OWL-S (OWL-S 1.0, OWL-S 0.9, DAML-S 0.7) descriptions. Data structures in the API have been designed closely to match the definition in the OWL-S ontology [28]. The API is incomplete in the areas where the OWL-S specification is incomplete, such as the definition of preconditions and effects. The names for packages, interfaces and methods in Java are chosen to match the names of ontologies, classes and properties in OWL-S ontologies.

OWL-S API is tightly coupled with Jena. We use them to load requests and advertisements using the OWL-S format. The OWL-S parsers, implemented in Java, extract inputs, outputs, preconditions and effects from requests and advertisements. These parameters are then sent to the reasoner Pellet to be classified.

2.3.4. WordNet

WordNet is a semantic lexicon for the English language (A semantic lexicon is a dictionary of words labeled with semantic classes so associations can be drawn between words that have not previously been encountered) [29]. It groups English words into sets of synonyms called synsets, provides short, general definitions, and records the various semantic relations between these synonym sets. The purpose is twofold: to produce a combination of dictionary and thesaurus that is more intuitively usable, and to support automatic text analysis and artificial intelligence applications [71]. WordNet distinguishes between nouns, verbs, adjectives and adverbs because they follow different grammatical rules. While semantic relations apply to all members of a synset, because they share a meaning but are all mutually synonyms, words can also be connected to other words through lexical relations, including antonyms (opposites of each other) and derivationally related, as well. WordNet also provides the polysemy count of a word: the number of synsets that contain the word. If a word participates in several synsets (i.e. has several senses) then typically some senses are much more common than others. WordNet quantifies this by the frequency score: in which several sample texts have all words semantically tagged with the corresponding synset, and then a count provided indicating how often a word appears in a specific sense [29].

There may be duplicate annotations of the same entities in different ontologies. Subsumption based ontology scoring will result in 0 point in these situations. The goal of using WordNet in PE matching is to detect concepts that are not in the same ontology definition but likely refer to the same entity. This provides very basic level of similarity assessment across heterogeneous ontologies [47].

2.3.5. WordNet Similarity

WordNet::Similarity is a freely available software package that makes it possible to measure the semantic similarity and relatedness between a pair of concepts (or synsets) [30]. It provides six measures of similarity, and three measures of relatedness, all of which are based on the lexical database WordNet. Measures of similarity use information found

in an is–a hierarchy of concepts (or synsets), and quantify how much concept *A* is like (or is similar to) concept *B*. For example, such a measure might show that an *automobile* is more like a *boat* than it is a *tree*, due to the fact that *automobile* and *boat* share *vehicle* as an ancestor in the WordNet noun hierarchy. WordNet is particularly well suited for similarity measures, since it organizes nouns and verbs into hierarchies of is–a relations [72]. Is–a relations in WordNet do not cross part of speech boundaries, so similarity measures are limited to making judgments between noun pairs (e.g., *cat* and *dog*) and verb pairs (e.g., *run* and *walk*). While WordNet also includes adjectives and adverbs, these are not organized into *is–a* hierarchies so similarity measures can not be applied.

WordNet:: Similarity module is our access point to WordNet similarity scores from an application point of view.

2.4. Evaluation Methods

In this section we will describe widely used evaluation methods for evaluating information retrieval performance of IR systems. Then, we will give an overview of Mann-Whitney significance test which we use to test the validity of the results obtained from experimental results.

2.4.1. IR Evaluation

The goal of an information retrieval (IR) system is to present a set of relevant documents to the user. Relevant documents are those documents that satisfy the user's information need. The concepts of recall and precision are central in most evaluation measures [74]. Recall is the fraction of relevant documents that is retrieved. Precision is the fraction of retrieved documents that is relevant [75].

$$recall = \frac{number\ of\ relevant\ documents\ retrieved}{number\ of\ relevant\ documents\ in\ collection} \tag{2.1}$$

$$precision = \frac{number\ of\ relevant\ documents\ retrieved}{total\ number\ of\ documents\ retrieved} \tag{2.2}$$

Both recall and precision are set-based measures; they operate on a fixed set of retrieved documents. Usually, as the set of retrieved documents get larger, recall grows at the cost of precision. For evaluating ranked lists of results, it is common to measure precision at different recall levels [32, 76]. One way to do this is by measuring precision (and recall) at fixed ranks. For example after 10, 20, and 30 retrieved documents (P@10, P@20, and P@30). Alternatively, precision can be measured after each relevant document that is retrieved. Suppose four relevant documents exist in the collection for a given information need and three of them are retrieved at ranks 1, 4, and 7. Then, the precision values at these levels are respectively P@1 = 1 (1/1), P@4 = 0.5 (2/4), and P@7 = 0.43 (3/7). The average of the precision values after each relevant document that is retrieved is called the (non-interpolated) average precision. To calculate it, the precision value of relevant documents that are not retrieved is assumed to be 0, thus the average precision in the given example is (1 + 0.5 + 0 .43) / 4 = 0.48.

It is good practice to evaluate information retrieval systems on more than a single query and to report averages over all queries [73]. A commonly used single measure is the mean average precision (MAP), the mean of the average precision values over all topics in the evaluation set. Another way of presenting multiple query results is with a recall-precision graph. A recall-precision graph plots precision against recall and is constructed by computing interpolated precision at 11 fixed recall points (0 to 1 in steps of 0.1) [74]. The interpolated precision at recall level l is defined as the maximum precision for any recall level greater than l. A graph for multiple queries is constructed by averaging the 11-point precision values over all queries. MAP and recall-precision graphs are the most commonly used measures for evaluation of IR systems.

In Figure 2.5, the precision is highest and the recall is lowest at the up left corner. At this point, the IR system returns relevant services but misses many useful ones. On contrary, the precision is lowest and the recall is highest at the lower right corner. Now, the IR system returns most relevant services but includes lots of junk. In order to have high values for both precision and recall, the graph should be near to the upper right corner.

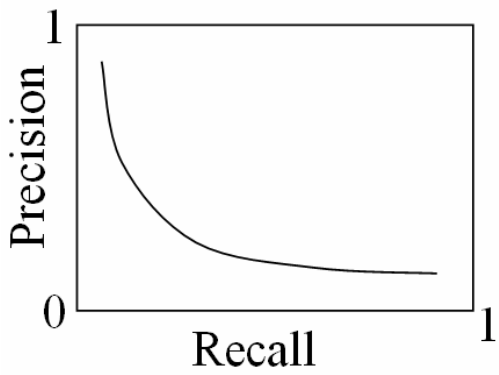

Figure 2.5. Recall-precision graph

We use the above mentioned IR evaluation methods to evaluate our proposed matchmaker. We compare the values gathered from the SAM and our proposed matchmaker to determine which one has higher information retrieval performance.

2.4.2. Mann-Whitney Test

A large body of statistical methods is available that comprises procedures not requiring the estimation of the population variance or mean and not stating hypotheses about parameters [31]. These testing procedures are termed nonparametric tests. As these methods also typically do no make assumptions about the nature of the distribution (e.g., normality) of the sampled populations (although they might assume that the sampled populations have the same dispersion or shape), they are sometimes referred to as distribution-free tests.

Mann and Whitney enlarged the original test (1947) proposed, for equal sample sized, by Wilcoxon (1945) [52, 77]. This test procedure is thus called the Wilcoxon-Mann-Whitney test, or, more commonly, Mann-Whitney test. For this test, as for many other nonparametric procedures, the actual measurements are not employed, but we use instead the ranks of the measurements. The data may be ranked either from the highest to lowest or from the lowest to highest values [31]. Mann-Whitney statistic is calculated as,

$$U = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1 \tag{2.3}$$

Where n_1 and n_2 are the number of observations in samples one and two respectively, and R_1 is the sum of the ranks of the observations in sample one. For the two-tailed hypotheses, the calculated U is compared with the two-tailed value of $U_{\alpha(2),n1,n2}$ found in the table of Mann-Whitney critical values. U' is calculated as,

$$U' = n_1 n_2 - U \tag{2.4}$$

Then, if either U or U' is as great as or greater than $U_{\alpha(2),n1,n2}$, null hypothesis is rejected at the α level of significance. For one-tailed hypotheses we need to declare which tail of the Mann-Whitney distribution is of interest, as this will determine whether U or U' must be calculated. This consideration is presented in Table 2.3 [31].

Table 2.3. The appropriate test statistic for the one-tailed Mann-Whitney test [31]

	H_0: Group 1 ≥ Group 2 Ha: Group 1 < Group 2	H_0: Group 1 ≤ Group 2 Ha: Group 1 > Group 2
Ranking done from low to high	U	U'
Ranking done from high to low	U'	U

We use the Mann-Whitney significance test in our evaluation to show that our results did not occur by chance.

The aim of this section was to make readers familiar with the idea and concepts of Semantic Web services and to present the most recent technology developments. Commencing from the vision and arising challenges for Semantic Web Services, this section explained in detail the most prominent technologies for Semantic Web Services and introduced techniques to evaluate the information retrieval performance and validity of a matchmaker.

3. RELATED WORK

The semantic matchmaking process is based on ontology formalizations over domains. In this 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 [32]. Advertisement denotes the service description of the existing services while the request indicates the picture of service requirements [33].

IO matching and PE matching show differences due to their natures. IO attributes describe syntactically which inputs are required by a service to function, and which outputs are returned; whereas PE rules describe under which conditions the service can be executed and the changes in the world after the execution of the service.

Traditional approaches to modeling semantic similarity between Web Services compute subsume relationship for function parameters in service profiles within a single ontology. Relatively less works exist on PE concerning matching. Paolucci et al., presents a matchmaking agent that uses subsumption relations built on DAML-S [34]. The basic idea is that an advertised service profile matches a requested service profile if (and only if) for each input parameter of the advertised profile there is a matching input parameter in the requested profile; and that for each output parameter in the requested profile there is a matching output parameter from the advertised profile. At parameter level, there are three levels of matches based on the relationship between the types of the pair of matching parameters: exact match, plugIn match, and subsumes match. The matching algorithm is limited to comparing input and output annotation of the advertisements with the requests. Li L. and Horrocks take a similar approach but from a different perspective [42]. The degree of match is determined by the relationship between the advertised concept A and the requested concept R. The degree of match is an exact match if $A = R$; plugIn match if $A \supset R$; subsumes match if $A \subset R$; intersection match if $A \wedge R$ is satisfiable; otherwise, disjoint. Aversano et al. presents a matchmaking algorithm utilizing service composition

based on input and output annotations of the services [35]. Yao et al. proposes a precondition and effect matching algorithm for services described in OWL-S. The algorithm is based on subsumption relationships; however it is described very intuitively [36]. They apply a discrete scale matching. During the matching process of B. Dong-Wei et al., firstly DL subsumption reasoning method is used to get the coarse set of services, and then more refined semantic distance calculation is made to improve the services distinguish capability based on input and output annotations [37]. Shen et al. designs a semantic ranking MSC to rank the results of advertisements matchmaking [38]. 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). Guo et al. describes capability matchmaking as matching inputs, outputs, preconditions and effects of services [39]. 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. PE matching using SWRL is given a future work. H. Wang and Z. Li propose a method for PE matching based on description logic SHOIN+(D) [40]. The preconditions and effects are described as knowledge bases, and the algorithm is based on knowledge base entailment. This method groups the matchmaking results in four categories: exact match, perfect match, side-effects match, and common match. This algorithm is not suitable for automatic matchmaking since it results in just a categorization of the match results and also not designed for Web services described in OWL-S. In addition to these references, there are many matching related works done based on WSMO ontologies, which are out of scope of this research [41]. In this research, we extend the previous matchmaking agent, SAM, to support PE matchmaking [43]–[45].

4. PROBLEM STATEMENT

As discussed in Section 2.2.1, service profile is the component in an OWL-S service description that is of particular interest during matchmaking of Semantic Web services. A service profile (service description) may have a set of parameters: the input parameters, output parameters, precondition parameters, and effect parameters. Actually, IO and PE descriptions reflect different aspects of capabilities. Input and output descriptions specify syntactic (or functional) information while precondition and effects descriptions reflect the semantics. The preconditions describe under which conditions the service could be executed and the effects reflect the changes of the world caused by the execution of the service. For example, in a sales transaction a selling service may require a valid credit card as a precondition, and the effect may be that the card is charged.

A service profile may service one of the two purposes: to describe an advertised service or to describe a requested service. Roughly, service matching is the process of comparing two service profiles, one advertised service profile and one requested service profile, to determine if the advertised service may fulfill the request. Service matching generally involves parameter pairing, and parameter matching. Parameter matching determines whether (and to what degree) two given parameters match; and parameter pairing associates the parameters from the two service profiles. Parameter matching is the basis for parameter pairing and both processes are mainly based on the parameter types.

As the number of available Web services on the Internet increases, finding a suitable Web service satisfying the needs becomes more difficult. Currently, Web services are described by WSDL (Web Service Definition Language) [2] and published on UDDI (Universal Description, Discovery, and Integration) [12, 13] registries. UDDI allows a keyword-based taxonomy for categorizing Web services. This categorization does not help much when there are a lot of matching Web services that fulfill a given keyword criteria. So, dynamic matchmaking is impossible by just using standard UDDI based approaches [80].

Historically matching has been done syntactically, the structure was the most important element of matching. There were many drawbacks of this approach: imposition of common standards was not successful and meaning of the compared properties of the services was lost [81]. This led to the evolution of semantic matching where matching is done based on the meanings and relations of the entities, namely according to the ontology definitions. Ontology is a data model that represents a set of concepts within a domain and the relationships between these concepts [82].

The problem that we focus on in this research is to find suitable Web services that satisfy a certain request by applying a semantic matchmaking process on run time. Some service matching algorithms are proposed to solve this problem. However, these algorithms mainly focus on the semantic similarities between advertisements and requests containing only inputs and outputs. They did not take into account the world states related to services, namely precondition and effect descriptions. There is lack of effective methods to solve the matchmaking of PE. The reason is complex, partly because of the IO is comparatively easy to cope with and partly because of the techniques to dealing with PE is not as mature as IO [40].

5. PROPOSED SOLUTION

In this research, we aim to provide an efficient and accurate precondition and effect matching algorithm, using scoring and ranking based on similarity distance information, extended subsumption, WordNet and property level similarity assessment. So before trying to perform the service, we can know whether it is indeed executable, i.e., whether all preconditions are satisfied, and whether applying it will achieve the desired effect, i.e., whether the effects are achieved or not. The experimental research so far has shown that simple subsumption based matchmaking is not sufficient to capture semantic similarity [55, 56]. Our main motivation is to capture semantic similarity between services in a more efficient way and eliminate false negatives during the matching. The main contribution of our algorithm is that it provides ordered ranking based solution for peculiar needs of PE matchmaking on OWL-S documents written in SWRL.

We consider service input, output, precondition and effect parameters and perform the matchmaking considering the IOPE. We have added PE matching capability of the previous research [11]. Thus, we will present the PE matching parts of the algorithm in this research in more detail.

The proposed matchmaker is capable of retrieving matching advertisements for a request by making all computations on run time. The matching is done for each request separately. PE matching is made based on parameters of the PE rules; and no rule engine is used to infer new information from these rules.

While searching for matching service advertisements for a request in terms of PE, different matching degrees are possible. Matching can be exact which means the request and the advertisement are the same, or it may not be exact but sufficient and still can be run without any problem. The problem in PE matchmaking is to define what is sufficient. We call a matching sufficient, if the advertisement and the request has some measurable degree of similarity. We apply a scoring based ranking scheme to assign scores to the advertisements and then order them from higher degree of match to lower degree of match.

In a dynamic composition the matching with the highest degree can be chosen automatically, or in a semi dynamic composition other mechanisms can be used together with ranking results to choose the advertisement to be invoked.

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

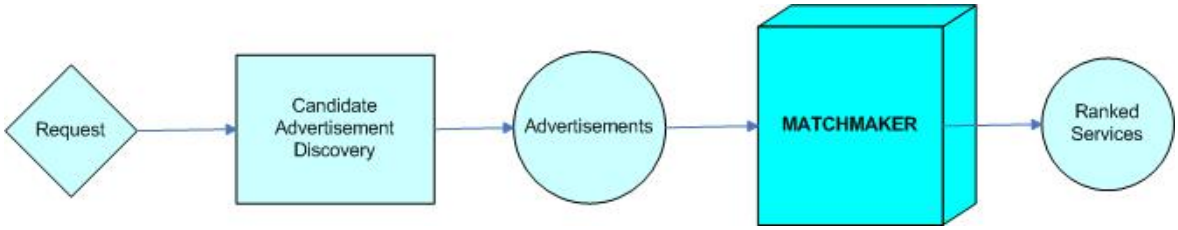

Figure 5.1. Matchmaking process

Figure 5.2 depicts the interactions between the modules of our matchmaking agent. Request service definition and the corresponding advertisements, which are discovered through conventional discovery mechanisms, are presented as inputs to the system. The matchmaking agent first reads the request and the advertisements into the memory. Then it uses the methods of scoring module to calculate similarity score for each parameter pair for a certain request-advertisement pair. Score calculations are done for input, output, precondition and effect descriptions separately. Score is calculated with the subsumption methods of the Pellet reasoner and with the Wordnet::similarity module. An ontology repository is used in subsumption methods and for distance weight assignments. These scores are then given to the bipartite graph module to calculate the maximum cardinality maximum weight matching. The matchmaker then adds and sorts the overall results (IOPE) and ranked service set is given as the output.

Scoring module is 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. Our proposed matching model uses the same

matchmaking architecture of the SAM. 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.

In order to match the PE definitions of the request and advertisements, we have enhanced the matchmaker and the scoring modules of the SAM shown in Figure 5.2. The new matchmaker module is able to parse PE annotations of the request and advertisements. Then, the scoring module calculates the matching scores for these annotations. This process will be explained in detail in the next section.

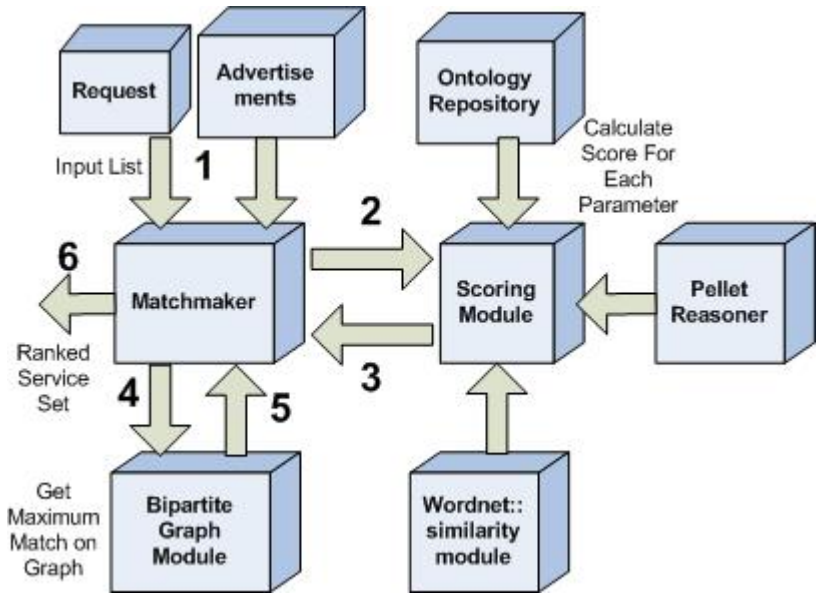

Figure 5.2. Matchmaking agent components

The main software components of our proposed matchmaking agent are shown in Figure 5.3. The top layer represents our IOPE enabled matchmaker. 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.

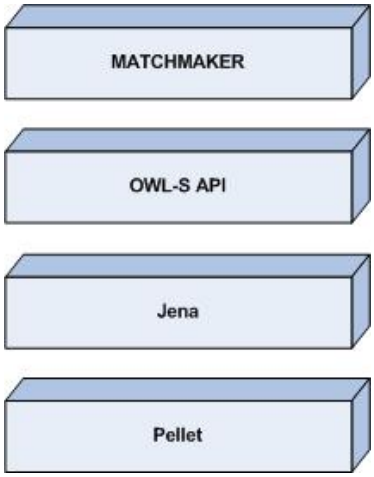

Figure 5.3. Software components of matchmaking agent

5.1. Matching Algorithm

Main skeleton of the matching algorithm is given in Table 5.1. Firstly, the algorithm loads the ontology definitions of the request and the advertisements into the memory. These definitions are used by the reasoner while determining subsumption relationships. Then input, output, precondition, and result scores are calculated for each advertisement. Lastly, the scores summed, sorted and returned as the output.

Table 5.1. Matching algorithm

```
sortedList matchingAdvertisements(req) {

        loadRequestOntology(req);
        loadAdvertisementOntologies(advertisementList);

        createInputBipartiteGraphs();
        createOutputBipartiteGraphs();
        createPreconditionBipartiteGraphs();
        createResultBipartiteGraphs();

        return sort(IOPEMatchList);
}
```

Input matching process is described in Table 5.2. Firstly, the algorithm reads the input definition of the request. Then for each advertisement, the inputs of the advertisement are compared with the inputs of the request. Subsumption based scoring

with semantic distance is used together with the WordNet scoring if it is enabled. The scores between the request and the advertisement pairings are used as the weights of the bipartite graph. We simply omit the services that require more inputs than the request has. We think that, these services cannot be executed successfully with insufficient number of inputs. The result of the scoring is obtained by calculating maximum cardinality maximum weight matching of the bipartite graph. This score becomes the input score of the advertisement and it is added to the input score list. Output scoring is calculated similarly, but we omit the services that have lesser number of outputs than the request.

Table 5.2. Input matching algorithm

```
createInputBipartiteGraphs() {

        reqInputs[reqSize] = readInputs(req);

        forall adv in advertisementList do {

                advInputs[advSize] = readInputs(adv);
                bipartiteGraph(reqSize, reqSize);

                if (reqSize < advSize) then {
                        inputResultList.add(adv, 0);
                        continue;
                }
                else {
                        forall i in reqInputs
                                forall j in advInputs do {

                                inputSubsumptionScore = subsumptionScore(i, j);
                                inputScore = inputSubsumptionScore * 0.9;

                                if(wordnetEnabled) then {
                                        inputWordnetScore = getWordNetScore(i, j);
                                }

                                inputScore += 0,1 * inputWordNetScore;
                                bipartiteGraph.setWeight(i, j, inputScore);
                        }

                        bipartiteMatchResult = bipartiteGraph.getMatching();
                        inputResultList.add(adv, bipartiteMatchResult);
                }
        }
}
```

Table 5.3. Precondition matching algorithm

```
createPreconditionBipartiteGraphs() {
        reqPreconditions[reqSize] = readPreconditions(req);

        forall adv in advertisementList do {
                advPreconditions[advSize] = readPreconditions(adv);
                bipartiteGraphPrecondition(reqSize, reqSize);
                if (reqSize < advSize) then {
                        preConditionResultList.add(adv, 0);
                        continue;
                }
                else {
                        forall i in reqPreconditions
                                forall j in advPreconditions do {

                                reqAtomList[reqAtomListSize] = i.getBody();
                                advAtomList[advAtomListSize] = j.getBody();
                                bipartiteGraphAtomList(reqAtomListSize, reqAtomListSize);
                                if (reqAtomListSize < advAtomListSize) then {
                                        bipartiteGraphPrecondition.setWeight(i, j, 0);
                                        continue;
                                }
                                forall k in reqAtomList
                                        forall l in advAtomList do {

                                        reqAtoms[reqAtomSize] = k..getAtoms();
                                        advAtoms[advAtomSize] = l.getAtoms();
                                        bipartiteGraphAtom(reqAtomSize, reqAtomSize);

                                        forall m in reqAtoms
                                                forall n in advAtoms do {

                                            preconditionSubsumptionScore = subsumptionScore(m, n);
                                            preconditionScore = preconditionSubsumptionScore * 0.9;

                                            if(wordnetEnabled) then {
                                                preconditionWordnetScore = getWordNetScore(i, j);
                                            }

                                            preconditionScore += 0,1 * preconditionWordNetScore;
                                            bipartiteGraphAtom.setweight(m, n, preconditionScore);
                                        }
                                        bipartiteAtomResult = bipartiteGraphAtom.getMatching();
                                        bipartiteGraphAtomList(k, l, bipartiteAtomResult);
                                }
                                bipartiteAtomListResult=norm(bipartiteGraphAtomList.getMatching());
                                bipartiteGraphPrecondition(i, j, bipartiteAtomListResult);
                        }
                        bipartiteMatchResult=norm(bipartiteGraphPrecondition.getMatching());
                        preconditionResultList.add(adv, bipartiteMatchResult);
                }
        }
}
```

Precondition/ effect matching is more complicated than input/ output. Preconditions are decomposable into atom lists and atom lists are decomposable into atoms. Atom lists are horn clauses like “Customer {hascard} VisaCard”, and the atoms are like “Customer” and “VisaCard”. Precondition and effect matching are done similarly. During the matching process multiple bipartite graphs are constructed, and score is calculated from atoms to preconditions in a backward fashion as in Table 5.3.

The algorithm reads each advertisement and compares the preconditions with the preconditions of the request. If number of preconditions in the request is less than the advertisement, zero score is assigned to the advertisement and skipped to the next service. We assume that the preconditions of the advertisement cannot be satisfied with the ones of the request. Then, we parse every precondition to get the atom lists. Precondition pairing is done using bipartite graph matchings between the atom lists. If the atom list of a request has lesser number of members than the advertisement, we assign zero score for the precondition pairing. In a similar fashion, we parse the atom lists to get the atoms and create bipartite graphs for the pairings. Using subsumption based scoring with semantic distance together with the WordNet scoring, if enabled; we assign scores for each atom pairings. Maximum weight atom score constitutes the score of the atom list pairings, and maximum weight atom list score constitutes the score of the precondition matching, and final scoring gives the overall precondition score of the advertisement. Scoring procedure is similar in result matching with the precondition matching. We assign zero score for the advertisements which have more results than the request, since invoking these services will bring side effects.

It is possible that the precondition/ result or internal atom list parameter count of the request does not match the parameter count of the advertisement. A preprocess step is introduced in the matchmaking process to add dummy parameters to the request or service in the following cases: If the advertisement precondition/ result/ atom list parameter count is less than the request precondition/ result/ atom list parameter count an extra parameter is added to the request precondition/ result/ atom list.

For the above cases, we introduced additional parameters to advertisement or request to make the parameter count equal and to be able to support perfect bipartite matching. We

ignored the additional parameters after the matchmaking process. However, we normalize the service similarity score. The idea is to apply a fair matchmaking so that a advertisement that can match the request with less precondition or result does not get a lower score due to its parameter count. So, we normalize the score of such a service using the following equation:

$$S_{new} = |P_r|/|P_s| * S_{old} \quad (5.1)$$

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.

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 atoms and the advertisement provides the parameter address, we may not capture the exact match between parameter pairs without such decomposition.

After all of the advertisements in the candidate set are evaluated against the request, the matchmaker stores the following information for each service and request pair: input matching score, output matching score, precondition matching score and result matching score.

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

$$S_{final} = w_{input} * S_{inputs} + w_{output} * S_{outputs} + w_{precondition} * S_{preconditions} + w_{result} * S_{results} \quad (5.2)$$

In the above equation, S_{inputs} , $S_{outputs}$, $S_{preconditions}$, and $S_{results}$ represent the similarity score considering only the input parameters, output parameters, preconditions and results respectively. w_{input} , w_{output} , $w_{precondition}$, and w_{result} represent the weights for the input, output, precondition and effect similarity scores, where they are fixed to 0.2, 0.3, 0.2 and 0.3. We prioritize output and effect matching since we think that the expected output and effect of a service is more important from a user's perspective. Alternatively, we could fix these coefficients equally, and the user or the service composer could determine them while requesting a service. Finally, S_{final} represents the final score of similarity considering all of the parameters.

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

$S_{x,y,}$ in the above equation, represents final similarity score between concepts *x* and *y*. $Subsumption_{x,y}$ represents the semantic score obtained through subsumption, property level matching and semantic distance. $WordNet_{x,y}$ represents the WordNet score for concept names. Since the concepts from the same ontology have greater probability of matching with respect to the concepts from heterogeneous ontologies, we promote the former ones during the scoring process. Thus, we have fixed the coefficients for subsumption and WordNet at 0.9 and 0.1, respectively. In the following parts we describe the components of our scoring algorithm.

5.1.1. Subsumption based Similarity Assessment

In OWL, two concepts have a subsumption relationship if there is a subset relationship between the set of objects described by two concepts. PE matching algorithm is tightly based on subsumption relationships. During matching, PE arguments of the advertisements and the request are compared with each other. This comparison is made according to the concepts (classes) of the arguments. If two concepts are stated to be complement or disjoint, zero score is directly assigned. Otherwise, we check for the subsumption relation and assign a score in the range [0, 1] as described in Table 5.4.

Table 5.4. Subsumption based score assignments

Relationship	Description	Score
Exact	Request and advertisement are equivalent	1.0
Plug-in	Request is subclass of advertisement = advertisement subsumes request	0.6 for preconditions 0.4 for effects
Subsume	Advertisement is subclass of request = request subsumes advertisement	0.4 for preconditions 0.6 for effects
Fail	No subsumption relation between request and advertisement exists	0

For the precondition example given in Section 2.2.2, we have the arguments *CustomerNum* and *VisaCard*. Suppose that these are the arguments of the precondition in a request and an advertisement precondition has the arguments *CustomerNum* and *CreditCard*. During the matching process each argument of the precondition in the request will be matched with every argument in advertisement precondition. Supposing only one precondition exists both in request and advertisement, matches in Figure 5.4 will be found with subsumption relation. Of course algorithm will compare *CustomerNum* with *CreditCard*, and *VisaCard* with *CustomerNum* and fail relationship will be attached for these comparisons.

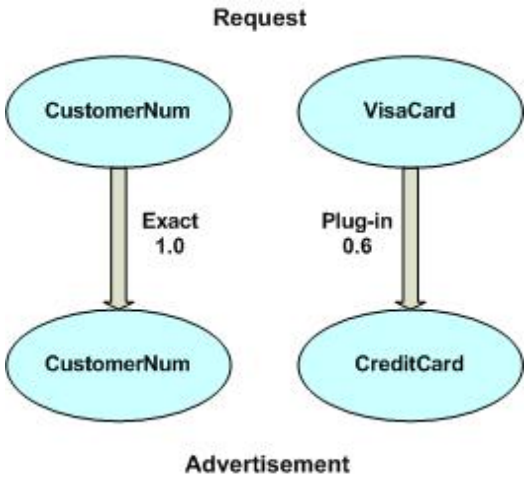

Figure 5.4. Subsumption based scoring results

5.1.2. Semantic Distance Based Scoring

Semantic distance is a rate of similarity of concepts. It is used as a measurement of semantic similarity between concepts [46]. Two kinds of distance calculation model exist: network distance model and information theoretic model. In information theoretic model semantic distance computes the information content of concepts and attaches the value to

ontology [79]. In network distance model, edge count distance is used assuming that edges between concepts represent uniform distance [36]. In our matchmaking agent we use network distance model. Each concept pair having subsume relationship in an ontology is assigned a weight in the range [0, 1]. In our implementation total weight of all subclasses of a parent concept must be equal to 1. This provides homogeneous distribution of the weights as we traverse the ontology from up to the bottom. Semantic distances can be assigned by ontology managers manually during the development of the ontologies. If not manually assigned, all direct subclasses of a parent class will have the same distance weight according to formula 5.4, where *x* is the parent class and *y* is a direct subclass of *x*.

$$dweight_{x,y} = \frac{1}{(numberofdirectsubclasses_x)} \tag{5.4}$$

The similarity between two concepts in an ontology decreases as the distance between them increases. Thus, to find semantic distance weight between any two concepts x and y in an ontology, we multiply the similarity distance values on the path from *x* to *y* (5.5). Semantic distance weights are used together with subsumption scores as given in (5.6).

$$dweight_{x,y} = dweight_{x,t} * dweight_{t,k} * ... * dweight_{...,y} \tag{5.5}$$

$$subweighteadscore = subsumptionscore * dweight \tag{5.6}$$

Since *VisaCard* is direct subclass of *CreditCard* then the score will be 0.6 * 0.5 = 0.3 according to the ontology definition in Figure 5.5.

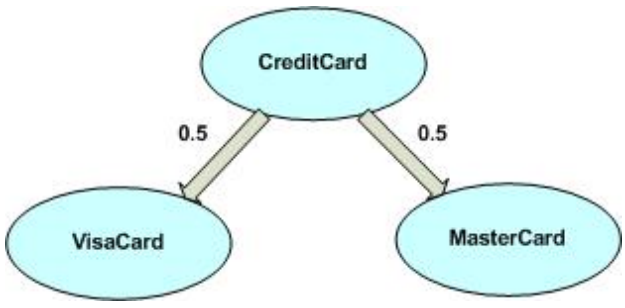

Figure 5.5. Semantic distance weight assignments

5.1.3. WordNet Based Scoring

While matching arguments of PE rules, using WordNet scoring is optional. We use Wordnet::Similarity project to assess similarity scores [30]. WordNet scores ranges in [0, 1]. This score is added to the score gathered from subsumption with semantic distance (5.7). A weight is used to regulate the effect of WordNet in matching score. Currently it is set to 0.1.

$$totscore = subweighteds core + (wordnetweight * wordnetscore) \quad (5.7)$$

5.1.4. Bipartite Graph Matching

Matching problems are often concerned with bipartite graphs. Since the IOPE parameters of a request and an advertisement constitute two disjoint sets, we can use bipartite graph matching for parameter pairing.

While matching the PE arguments of request and service we must pair them. In section 5.1.1. we paired the arguments as (CustomerNum, CustomerNum), and (VisaCard, CreditCard). In fact there were other pairings (CustomerNum, VisaCard), and (VisaCard, CustomerNum). To find the correct parameter pairing, the problem can be transformed into a bipartite matching problem and then a maximum bipartite matching algorithm can be applied to find the optimum parameter pairing. We use the Hungarian method for weighted bipartite matching since it is the best known fastest algorithm [48].

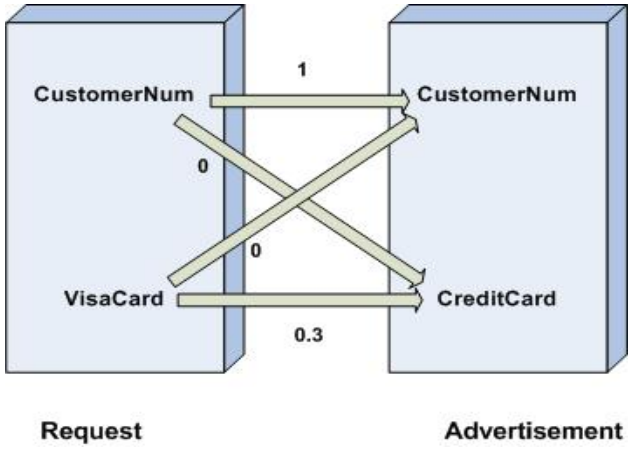

Figure 5.6. Maximum weight bipartite graph matching

Figure 5.6 shows the bipartite graph our example. Possible matches are {(CustomerNum, CustomerNum), (CustomerNum, CreditCard), (VisaCard, CustomerNum), (VisaCard, CreditCard)}. Maximum cardinality matchings are {(CustomerNum, CustomerNum), (VisaCard, CreditCard)} and {(CustomerNum, CreditCard), (VisaCard, CustomerNum)}. Finally, maximum cardinality maximum weight matching is {(CustomerNum, CustomerNum), (VisaCard, CreditCard)}.

Transforming parameter pairing problem to bipartite matching is harder in PE matching than IO matching. For example for input matching we can partition the request inputs and service inputs into two sets and draw an edge between each node weighted with the scores calculated as in previous sections. Then we can use the maximum cardinality maximum weight bipartite matching algorithms to find the maximum matching with maximum weight. But for precondition matching we cannot partition the arguments of the request and the service into two sets since arguments in the same predicate must be matched with another predicate's arguments together. Also, a precondition of a request must be compared with a precondition of an advertisement as a whole.

5.1.5. Condition Matching Algorithm Example

In this part we will describe our condition matching algorithm with an example to make it clear. We will try to match preconditions of one request with one advertisement. In a real case a request is matched with many advertisements and obtained scores are ordered.

Our example is a scenario taken from a sales transaction. Credit card and user authorization checks constitute basic control blocks of a sales transaction. Our example request service annotation declares that the customer has a valid VisaCard and a valid password in order to access the PurchaseZone of the service provider. On the other hand, the advertisement requires that the requester must have a valid credit card, and a valid password. Our example request and advertisement service are given in Table 5.5 and Table 5.6.

Table 5.5. Request example

Precondition	Predicate	Arguments
(a) CustomerHasVisaCard	hasCard	CustomerNum VisaCard
(b) CustomerHasAuthorization	hasPassword	CustomerNum Password
	hasAccess	CustomerNum PurchaseZone

Table 5.6. Advertisement example

Precondition	Predicate	Arguments
(c) CustomerHasCard	hasCard	CustomerNum Card
(d) CustomerHasAuthorization	hasPassword	CustomerNum Password

If the advertisement has more preconditions then the request, we assign zero score for the advertisement. Since a service cannot be run unless its preconditions are satisfied, we omit the services having more preconditions than the request. Similarly if an advertisement precondition has more predicates then the request precondition we assign zero score for that pre-condition pairing. In this example we firstly try to match *a* and *c*. They both have one predicate. Arguments of the predicates are paired (4 pairs) and assigned a score according to our scoring methods described in previous sections. Then a bipartite graph is constituted and maximum weight sum is calculated. This score is the matching score of *(a)* and *(c)*. Then we try to match *(a)* and *(d)*. Case is similar as in *(a)* and *(c)*, but *VisaCard* and *Password* will not semantically match, and the score will be less. While we match *(b)* and *(d)* it is the case that the request precondition has more predicates than the service precondition. In this matching we will use multiple bipartite graphs for maximum weight sum as in Figure 5.7. Maximum weight sums obtained from argument graphs are used as the weights of the predicate graph. Applying the maximum sum algorithm on predicate graph gives the score of *(b)* and *(d)*. Lastly we match *(b)* and *(c)* similar to *(b)* and *(d)*. Now we have scorings for (*a, c*), (*a, d*), (*b, c*), and (*b, d*) pairs. Last step of the algorithm

creates another bipartite graph from these pairs as in Figure 5.8 and Figure 5.9. Maximum weight sum of this graph gives the scoring of the advertisement.

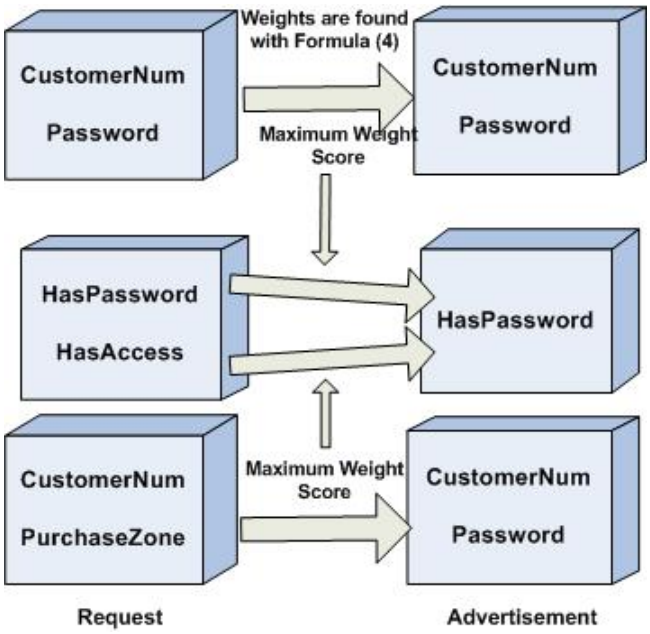

Figure 5.7. Argument and predicate matching

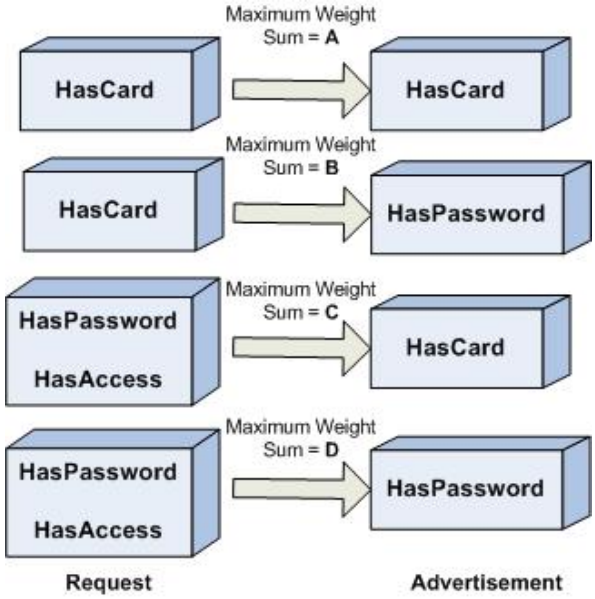

Figure 5.8. Predicate matching

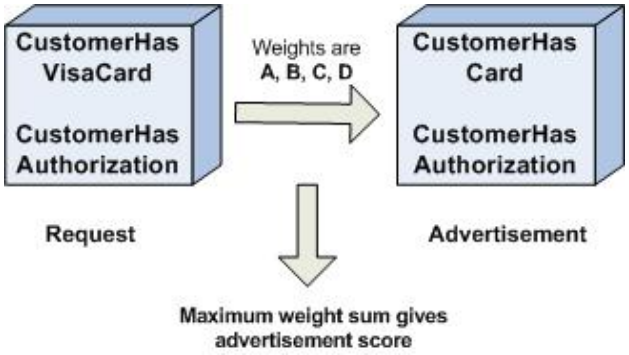

Figure 5.9. Precondition matching & advertisement score

In this section we have described the matching algorithm in detail. The ideal outcome of a semantic matchmaker is the correctly matching advertisement(s) given a request description. The definition of the correct matching may differ in different situations. We aim to provide the requester with the right set of advertisements in the result of matchmaking. We try to do this by decomposing the IOPE annotations of the services, and applying a scoring scheme. But, we cannot be sure whether we are supplying the requester with the right set of advertisements all the time. For example, a service may include an input parameter in its service profile, but may still work although this parameter is not supplied by the requestor. In such situations our matchmaker works with the assumptions aforementioned. Thus, we should evaluate our algorithm in different cases with different expectations, or outcomes where the right matching advertisements are known a priori. In the next section we will test out matchmaker to show whether it increases the ratio of finding correct advertisements in various test cases.

6. Experimental Results

6.1. Test Ontology

We used and extended the ontology definitions from "OWL-S Service Retrieval Test Collection version 2.1" (OWL-S TC) in our experiments [49]. For the requests and advertisement sets, we created PE enabled services making use of these ontology definitions. In order to evaluate the performance of our proposed matchmaking agent we extended the book ontology in OWL-S TC and also modified related request and advertisement definitions accordingly. 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. 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.

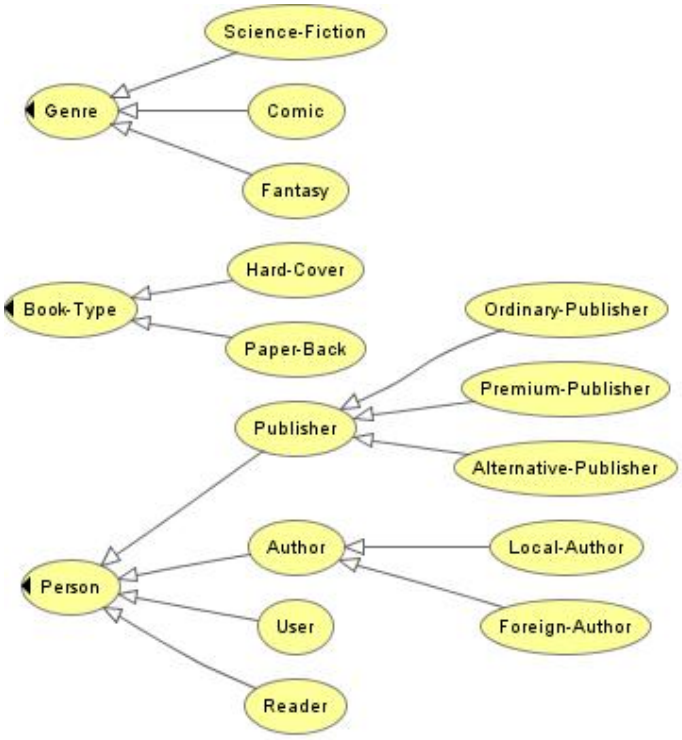

Figure 6.1. A section of book ontology

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.

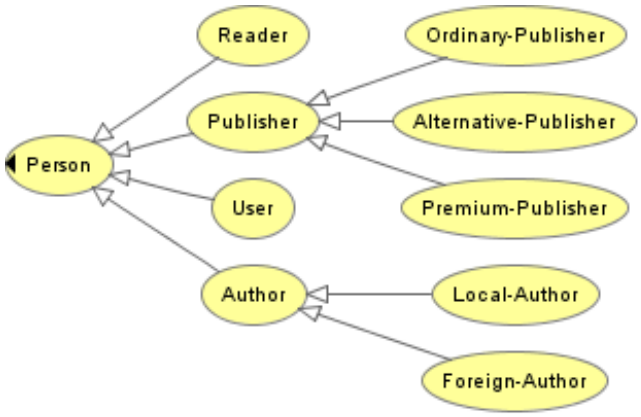

Figure 6.2. Person ontology section

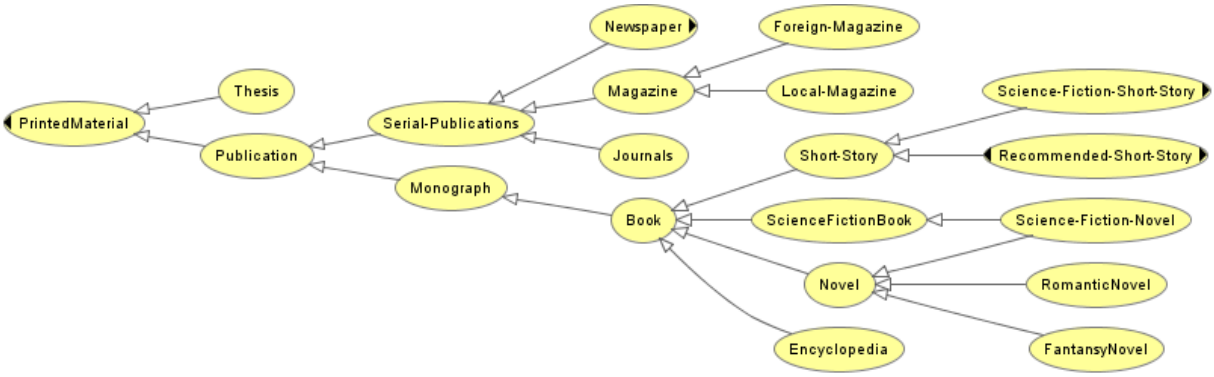

Figure 6.3. Printed Material ontology section

The OWL-S documents describing the services make use of the book and concepts ontology in OWL-S TC. Table 6.1 represents the header section of an OWL-S document demonstrating how these ontologies are imported. Table 6.2 presents the service section of an OWL-S document. Table 6.3 represents the profile section of an OWL-S document, describing how the inputs, outputs, preconditions and effects of the service are referred. Table 6.4 presents the process section of the OWL-S document and represents how inputs, outputs, preconditions and effects are defined.

Table 6.1. OWL-S document header

```
<?xml version="1.0" encoding="WINDOWS-1252"?>

<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-s/1.1/Grounding.owl">
  <!ENTITY expr "http://www.daml.org/services/owl-s/1.1/generic/Expression.owl">
  <!ENTITY swrl "http://www.w3.org/2003/11/swrl">
  <!ENTITY groundingWSDL "http://www.mindswap.org/axis/services/TranslatorService?wsdl">
  <!ENTITY factbook "http://www.daml.org/2003/09/factbook/languages">
  <!ENTITY mind "http://www.mindswap.org/2004/owl-s/1.1/MindswapProfileHierarchy.owl">
  <!ENTITY this "http://127.0.0.1/services/1.1/s11.owls">
]>

<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
  xmlns:expr="&expr;#"
  xmlns:swrl="&swrl;#"
  xmlns:factbook="&factbook;#"
  xmlns:mind="&mind;#"
  xml:base="&this;"
  xmlns="&this;#"
>

<owl:Ontology rdf:about="">
        <owl:imports rdf:resource="http://127.0.0.1/ontology/Service.owl" />
        <owl:imports rdf:resource="http://127.0.0.1/ontology/Process.owl" />
        <owl:imports rdf:resource="http://127.0.0.1/ontology/Profile.owl" />
        <owl:imports rdf:resource="http://127.0.0.1/ontology/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:imports rdf:resource="http://127.0.0.1/ontology/Mid-level-ontology.owl" />
        <owl:imports rdf:resource="http://127.0.0.1/ontology/SUMO.owl" />
</owl:Ontology>
```

Table 6.2. OWL-S document service section

```
<service:Service rdf:ID="S11_SERVICE">
        <service:presents rdf:resource="#S11_PROFILE"/>
        <service:describedBy rdf:resource="#S11_PROCESS_MODEL"/>
        <service:supports rdf:resource="#S11_GROUNDING"/>
</service:Service>
```

Table 6.3. OWL-S document profile section

```
<profile:Profile rdf:ID="S11_PROFILE">
        <service:isPresentedBy rdf:resource="#S11_SERVICE"/>
        <profile:serviceName xml:lang="en">
                Service 11
        </profile:serviceName>
        <profile:textDescription xml:lang="en">
                Service 11 description
        </profile:textDescription>
        <profile:hasInput  rdf:resource="#_PUBLICATION"/>
        <profile:hasInput  rdf:resource="#_BOOKTYPE"/>
        <profile:hasInput  rdf:resource="#_READER"/>
        <profile:hasOutput rdf:resource="#_REVIEW"/>
        <profile:hasOutput rdf:resource="#_FANTASY"/>
        <profile:hasPrecondition rdf:resource="#_HASCARD"/>
        <profile:hasResult rdf:resource="#_DELIVERY"/>
        <profile:hasResult rdf:resource="#_SEND"/>
        <profile:has_process rdf:resource="S11_PROCESS" />
</profile:Profile>
```

Table 6.4. OWL-S document process section

```
<process:ProcessModel rdf:ID="S11_PROCESS_MODEL">
        <service:describes rdf:resource="#S11_SERVICE"/>
        <process:hasProcess rdf:resource="#S11_PROCESS"/>
</process:ProcessModel>

<process:AtomicProcess rdf:ID="S11_PROCESS">
        <process:hasInput  rdf:resource="#_PUBLICATION"/>
        <process:hasInput  rdf:resource="#_BOOKTYPE"/>
        <process:hasInput  rdf:resource="#_READER"/>
        <process:hasOutput rdf:resource="#_REVIEW"/>
        <process:hasOutput rdf:resource="#_FANTASY"/>
        <process:hasPrecondition rdf:resource="#_HASCARD"/>
        <process:hasResult rdf:resource="#_DELIVERY"/>
</process:AtomicProcess>

<process:Input rdf:ID="_PUBLICATION">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
                http://127.0.0.1/ontology/modified_books.owl#Publication
        </process:parameterType>
        <rdfs:label></rdfs:label>
</process:Input>

<process:Input rdf:ID="_BOOKTYPE">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
                http://127.0.0.1/ontology/modified_books.owl#Book-Type
        </process:parameterType>
        <rdfs:label></rdfs:label>
</process:Input>

<process:Input rdf:ID="_READER">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
                http://127.0.0.1/ontology/modified_books.owl#Reader
        </process:parameterType>
        <rdfs:label></rdfs:label>
</process:Input>
```

Table 6.4. OWL-S document process section (continued)

```
<process:Output  rdf:ID="_REVIEW">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
                http://127.0.0.1/ontology/modified_books.owl#Review
        </process:parameterType>
        <rdfs:label></rdfs:label>
</process:Output>

<process:Output  rdf:ID="_FANTASY">
        <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
                http://127.0.0.1/ontology/modified_books.owl#Fantasy
        </process:parameterType>
        <rdfs:label></rdfs:label>
</process:Output>

<expr:SWRL-Condition rdf:ID="_HASCARD">
        <rdfs:label>hasCard(_User, _CreditCardAccount)</rdfs:label>
        <expr:expressionLanguage rdf:resource="&expr;#SWRL"/>
        <expr:expressionBody rdf:parseType="Literal">
        <swrl:AtomList>
        <rdf:first>
         <swrl:IndividualPropertyAtom>
        <swrl:propertyPredicate rdf:resource="http://127.0.0.1/services/1.1/s11.owls#HASCARD"/>
        <swrl:argument1 rdf:resource="http://127.0.0.1/ontology/modified_books.owl#User"/>
        <swrl:argument2 rdf:resource="http://127.0.0.1/ontology/Mid-level-
        ontology.owl#CreditCardAccount"/>
        </swrl:IndividualPropertyAtom>
        </rdf:first>
        <rdf:rest rdf:resource="&rdf;#nil"/>
        </swrl:AtomList>
        </expr:expressionBody>
</expr:SWRL-Condition>

<process:Result rdf:ID="_DELIVERY">
        <process:hasEffect>
        <expr:SWRL-Expression>
         <rdfs:comment>DELIVERY</rdfs:comment>
        <expr:expressionBody rdf:parseType="Literal">
        <swrl:AtomList>
        <rdf:first>
         <swrl:IndividualPropertyAtom>
        <swrl:propertyPredicate rdf:resource="http://127.0.0.1/services/1.1/s11.owls#DELIVERY"/>
        <swrl:argument1 rdf:resource="http://127.0.0.1/ontology/Mid-level-
        ontology.owl#AirTransportation"/>
        <swrl:argument2 rdf:resource="http://127.0.0.1/ontology/Mid-level-
        ontology.owl#Address"/>
        </swrl:IndividualPropertyAtom>
        </rdf:first>
        <rdf:rest rdf:resource="&rdf;#nil"/>
        </swrl:AtomList>
        </expr:expressionBody>
        </expr:SWRL-Expression>
        /process:hasEffect>
</process:Result>
```

6.2. Test Environment

Our matchmaker was developed in Eclipse IDE. We have implemented the matching algorithm in Java. We have used the OWL-S Java API to parse the OWL-S profiles, for reasoning operations, and for utilizing Jena constructs. OWL-S API is the most comprehensive library for working with OWL-S documents [28]. Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for OWL and includes a rule-based inference engine. We have used Pellet reasoner instead of Jena's native reasoner due to Pellet's superior capabilities [27].

In our experiments, we compared our results with that of SAM [43]–[45]. SAM has IO matching capability whereas our proposed model can do both IO and PE matching. For measuring the retrieval performance of two algorithms we created a test collection of 100 services described in OWL-S including PE annotations in SWRL. In order to create these services, we used ontology definitions in OWL-S TC as basis [49]. We assumed that these 100 services have been selected from UDDI registry with a keyword search so we did not include completely unrelated services in the test collection. Services are different from each other in terms of their IOPE annotations. We created 20 test queries (requests: *r1* to *r20*) (Table 6.5.) to evaluate the information retrieval performance of the two algorithms.

Table 6.5. Test queries

	Inputs	**Outputs**	**Preconditions**	**Results**
r1	ordinarypublisher	localauthor	hasAccount(User, Journals)	Delivery(AirTransportation, Address)
	novel	genre	hasCard(User, CreditCardAccount)	
	paperback			
r2	book	price	hasAccount(User, Journals)	Delivery(Transportation, Address)
r3	book	price	hasAccount(User, Journals)	Delivery(Transportation, Address)
			hasCard(User, CreditCardAccount)	
r4	author	price	hasAccount(User, Journals)	Delivery(Transportation, Address)
	book		hasCard(User, CreditCardAccount)	
	booktype			
	publisher			
r5	author		hasAccount(User, Journals)	Delivery(Transportation, Address)

Table 6.5. Test queries (continued)

	book		hasCard(User, CreditCardAccount)	
	booktype			
	reader			
	publisher			
r6	book	price	hasAccount(User, Journals)	Delivery(Transportation, Address)
			hasCard(User, CreditCardAccount)	Send(FinancialBill, Address)
r7	ordinarypublisher	localauthor	hasAccount(User, Journals)	Delivery(AirTransportation, Address)
	novel	genre	hasCard(User, CreditCardAccount)	Send(FinancialBill, Address)
	paperback			
r8	author	price	hasAccount(User, Journals)	Delivery(Transportation, Address)
	book		hasCard(User, CreditCardAccount)	Send(FinancialBill, Address)
	booktype			
	publisher			
r9	author		hasAccount(User, Journals)	Delivery(Transportation, Address)
	book		hasCard(User, CreditCardAccount)	Send(FinancialBill, Address)
	booktype			
	reader			
	publisher			
r10	author	price	hasAccount(User, Journals)	Delivery(Transportation, Address)
	book		hasCard(User, CreditCardAccount)	Send(FinancialBill, Address)
r11	ordinarypublisher	localauthor	hasCard(User, CreditCardAccount)	Delivery(AirTransportation, Address)
	novel	genre		
	paperback			
r12	book	price	hasCard(User, CreditCardAccount)	Delivery(Transportation, Address)
r13	author	price	hasCard(User, CreditCardAccount)	Delivery(Transportation, Address)
	book			Send(FinancialBill, Address)
	booktype			
	publisher			
r14	author	price	hasAccount(User, Journals)	Delivery(Transportation, Address)
	book			Send(FinancialBill, Address)
	booktype			
	publisher			
r15	ordinarypublisher	localauthor	hasAccount(User, Journals)	Delivery(AirTransportation, Address)
	novel	genre		Send(FinancialBill, Address)
	paperback			
r16	ordinarypublisher	localauthor	hasAccount(User, Journals)	Delivery(AirTransportation, Address)
	novel	genre		

Table 6.5. Test queries (continued)

	paperback			
r17	ordinarypublisher	localauthor	hasAccount(User, Journals)	Delivery(AirTransportation, Address)
	novel			Send(FinancialBill, Address)
	paperback			
r18	publisher		hasAccount(User, Journals)	Delivery(Transportation, Address)
	book		hasCard(User, CreditCardAccount)	Send(FinancialBill, Address)
r19	publisher		hasAccount(User, Journals)	Delivery(Transportation, Address)
	book		hasCard(User, CreditCardAccount)	
r20	publisher			Delivery(Transportation, Address)
	book		hasCard(User, CreditCardAccount)	

6.3. Test Results

Despite the amount of attention that Semantic Web service matchmaking receives, little effort is devoted to experimental and comparative evaluation of the approaches [50]. There exists only one publicly available large test collection of Semantic Web services, namely OWL-S test collection 2, but it does not include precondition and effect annotations. In terms of PE matching, to the best of our knowledge, there has been no effort by groups to work with the same realistically sized service sets, the same queries, and the same evaluation techniques. For these reasons, comparison of the results across different matching systems cannot be done objectively.

In order to evaluate our matchmaking algorithm we used classical information retrieval evaluation techniques. We adopted the evaluation strategy of micro-averaging the individual interpolated precision-recall curves [53]. Voorhees, E. M states that high correlations exist among the rankings of the systems produced using different relevance judgments sets, and increasing the number of judges makes no improvement in comparing the performances of two systems [51]. Thus, in our experiments one judge determined the relevant advertisement set for each query. We run both IO and IOPE algorithms separately for each 20 queries and determined the ranked retrieved advertisement set for each algorithm.

For evaluating ranked lists of matching results, it is common to measure precision at different recall levels [32, 76]. Taking the services determined by the judge as the relevant set, and the services resulted from the algorithms as the retrieved set, we calculated the precision values for each recall level for both algorithms separately. Then, we calculated interpolated precision values for every recall level *r* by taking the maximum precision for any recall level greater or equal to *r*. These calculations gave us interpolated precision values for 11 point recall levels (from 0 to 1) for both algorithms. Micro-averaging all the interpolated precision values in different recall levels; we determined the ultimate mean values of the test. We plotted interpolated mean precision values for fixed recall intervals for both algorithms as in Fig. 6.4. Both curves slope downwards from left to right, showing that as more relevant services are found (recall increases), more non-relevant services are retrieved (precision decreases). Since IOPE matching algorithm's curve is closer to the upper right-hand corner of the graph (where recall and precision are maximized) its information retrieval performance is better than IO matching algorithm. We found the micro-averaged interpolated precision of IO algorithm to be 0.173, and IOPE algorithm to be 0.552. This also shows the superiority of IOPE algorithm to IO algorithm in terms of information retrieval.

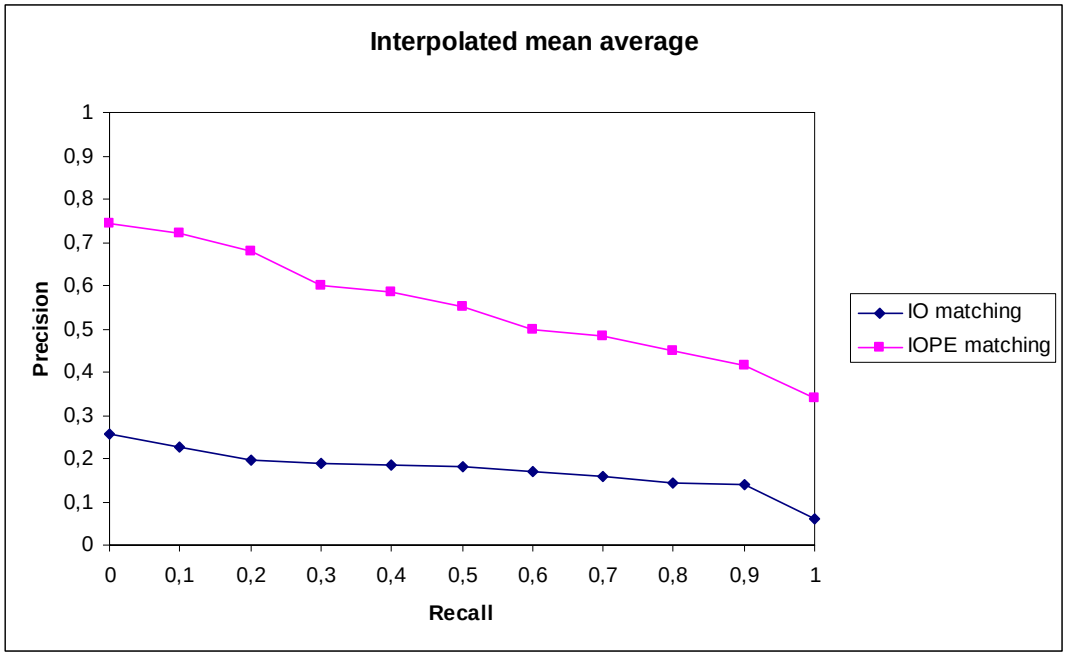

Figure 6.4. Interpolated mean average precision values for 11pt. recall levels

Table 6.6. Interpolated average precision values for the IO algorithm

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	Interpolated Average Precision
1	1.000	0.600	0.600	0.529	0.481	0.481	0.467	0.405	0.324	0.324	0.245	**0.496**
2	0.333	0.333	0.100	0.080	0.080	0.080	0.080	0.080	0.080	0.080	0.080	**0.128**
3	0.389	0.389	0.389	0.389	0.320	0.265	0.265	0.233	0.233	0.225	0.204	**0.300**
4	0.500	0.364	0.353	0.353	0.352	0.352	0.352	0.352	0.352	0.352	0.000	**0.335**
5	0.500	0.500	0.318	0.318	0.318	0.318	0.318	0.318	0.318	0.318	0.000	**0.322**
6	0.118	0.118	0.118	0.118	0.118	0.118	0.044	0.044	0.041	0.041	0.041	**0.083**
7	0.333	0.240	0.240	0.240	0.240	0.240	0.175	0.150	0.150	0.149	0.117	**0.207**
8	0.250	0.250	0.250	0.250	0.250	0.250	0.226	0.216	0.216	0.216	0.000	**0.216**
9	0.239	0.239	0.239	0.239	0.239	0.239	0.226	0.216	0.216	0.216	0.000	**0.210**
10	0.069	0.069	0.069	0.069	0.069	0.069	0.069	0.069	0.069	0.000	0.000	**0.056**
11	0.294	0.294	0.294	0.294	0.294	0.294	0.294	0.294	0.150	0.104	0.104	**0.247**
12	0.133	0.133	0.133	0.133	0.133	0.120	0.120	0.083	0.083	0.068	0.068	**0.110**
13	0.086	0.086	0.086	0.086	0.086	0.086	0.086	0.086	0.086	0.086	0.086	**0.086**
14	0.077	0.077	0.077	0.077	0.077	0.077	0.048	0.048	0.000	0.000	0.000	**0.051**
15	0.045	0.045	0.045	0.045	0.045	0.045	0.017	0.017	0.017	0.017	0.017	**0.032**
16	0.087	0.087	0.087	0.087	0.087	0.074	0.074	0.074	0.074	0.074	0.074	**0.080**
17	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031	0.031	**0.031**
18	0.091	0.091	0.091	0.091	0.091	0.091	0.085	0.085	0.085	0.085	0.085	**0.088**
19	0.500	0.500	0.300	0.300	0.300	0.300	0.300	0.300	0.300	0.300	0.000	**0.309**
20	0.080	0.080	0.080	0.080	0.080	0.080	0.080	0.080	0.080	0.080	0.080	**0.080**
Mean	**0.258**	**0.226**	**0.195**	**0.191**	**0.185**	**0.181**	**0.168**	**0.159**	**0.145**	**0.138**	**0.062**	**0.173**

Table 6.7. Interpolated average precision values for the IOPE algorithm

	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1	Interpolated Average Precision
1	1,000	1,000	1,000	1,000	1,000	1,000	0,875	0,833	0,833	0,733	0,714	**0,908**
2	1,000	1,000	1,000	1,000	1,000	0,625	0,625	0,625	0,292	0,292	0,292	**0,705**
3	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,556	0,556	**0,919**
4	1,000	1,000	1,000	0,923	0,900	0,900	0,900	0,900	0,900	0,886	0,780	**0,917**
5	1,000	1,000	1,000	0,900	0,800	0,538	0,538	0,538	0,538	0,538	0,518	**0,719**
6	0,333	0,333	0,333	0,333	0,333	0,333	0,114	0,114	0,114	0,114	0,114	**0,234**
7	1,000	0,500	0,500	0,500	0,500	0,500	0,391	0,391	0,391	0,345	0,262	**0,480**
8	1,000	1,000	0,615	0,615	0,615	0,611	0,600	0,600	0,576	0,576	0,526	**0,667**
9	1,000	1,000	0,600	0,600	0,600	0,579	0,365	0,365	0,365	0,365	0,357	**0,563**
10	0,200	0,200	0,200	0,200	0,200	0,200	0,200	0,160	0,160	0,143	0,143	**0,182**
11	0,625	0,625	0,625	0,625	0,625	0,625	0,625	0,625	0,269	0,269	0,269	**0,528**
12	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,238	**0,931**
13	0,333	0,333	0,333	0,171	0,171	0,171	0,171	0,171	0,171	0,171	0,171	**0,215**
14	1,000	1,000	1,000	0,400	0,400	0,400	0,091	0,091	0,087	0,087	0,087	**0,422**
15	0,250	0,250	0,250	0,250	0,250	0,250	0,250	0,250	0,250	0,250	0,080	**0,235**
16	0,250	0,250	0,250	0,222	0,222	0,185	0,185	0,185	0,185	0,185	0,185	**0,210**
17	0,053	0,053	0,053	0,053	0,053	0,053	0,053	0,053	0,053	0,053	0,053	**0,053**
18	1,000	1,000	1,000	0,333	0,333	0,333	0,286	0,286	0,286	0,286	0,286	**0,494**
19	1,000	1,000	1,000	1,000	0,882	0,882	0,882	0,882	0,882	0,882	0,593	**0,899**
20	0,846	0,846	0,846	0,846	0,846	0,846	0,846	0,593	0,593	0,593	0,593	**0,754**
Mean	**0,745**	**0,720**	**0,680**	**0,599**	**0,587**	**0,552**	**0,500**	**0,483**	**0,447**	**0,416**	**0,341**	**0,552**

Table 6.6 shows the interpolated average precision values of 20 requests for the 11 point recall values in the IO based algorithm, and Table 6.7 shows the same values for the IOPE based algorithm. IOPE algorithm shows superior information retrieval performance in all of the requests. It can also find relevant advertisements much earlier. Thus, the performance of IOPE matching algorithm is superior to IO matching algorithm at the precision, and for discovering Semantic Web services, precision is more important to user than the recall since we need to find a precise Web service as user specified to perform the specified task.

In order to show that these results did not occur by chance we applied one tailed Mann-Whitney significance test for interpolated average precision values. Since this test does not require normality of the sampled populations, and the two sample set values (IO and IOPE) are independent we can use it safely.

Our null and alternate hypotheses are given as;

H_0 : Precision of the IOPE matching algorithm is not greater than the precision of IO matching algorithm.

H_A : Precision of the IOPE matching algorithm is greater than the precision of IO matching algorithm.

Firstly, we rank the precision values for two algorithms from lower to higher, and assign rank values for combined sample set as in Table 6.8. Then, U' is calculated as 364 from the equation (6.1) and $U_{0.05(1),20,20}$ is read as 262 from the table of Mann-Whitney critical values. As 364 > 262 we reject H_0 at the 5% significance level. In fact the precision of the IOPE matching algorithm is highly significantly greater than IO algorithm with the significance level less than 1%.

$$U' = n_2 n_1 + \frac{n_2(n_2+1)}{2} - R_2 \qquad (6.1)$$

Table 6.8. Algorithm ranks

Query	IO Rating	IO Rank	IOPE Rating	IOPE Rank
1	0,908116883	36	0,49595845	28
2	0,704545455	32	0,12821317	11
3	0,919191919	38	0,30009117	21
4	0,917207183	37	0,33483228	24
5	0,719105894	33	0,32231405	23
6	0,233766234	18	0,08332424	7
7	0,480058672	26	0,20678258	13
8	0,6668269	31	0,21583112	17
9	0,56342079	30	0,20990227	15
10	0,182337662	12	0,05642633	4
11	0,527972028	29	0,24653604	20
12	0,930735931	39	0,11002906	10
13	0,215077605	16	0,08641975	8
14	0,422062522	25	0,05061605	2
15	0,234545455	19	0,03236915	1
16	0,20959596	14	0,07963264	5
17	0,052631579	3	0,03125	0
18	0,493506494	27	0,0882715	9
19	0,89879184	35	0,30909091	22
20	0,753949754	34	0,08	6
n_1=n_2=20		**R_1=534**		**R_2=246**

Table 6.9. Feature Comparison

Feature	IO algorithm	IOPE algorithm
Language	OWL-S	OWL-S and SWRL
Scope	OWL-S services with input or output annotations	OWL-S services with input or output or precondition or result annotations
Outcome	Ranked service list	Ranked service list
Runtime performance	Faster	Slower due to PE matching cost
Scalability	Very scalable	Very scalable
Complexity of descriptions required	Less complex	More complex due to SWRL
Level of guarantee	Lower guarantee	Higher guarantee (higher retrieval precision)

We also present a feature comparison table of the algorithms. Using the comparison criteria in Küster et al. we compared the two algorithms in Table 6.9 [50]. Both of the algorithms are written for the OWL-S language; however the IOPE algorithm allows PE definitions written in SWRL. Scope of the IO algorithm is narrower since it only handles IO definitions. Both algorithms have similar outcome: ranked service list. In terms of runtime performance, the IO algorithm is better than the IOPE algorithm, since processing

PE annotations takes time. Both algorithms have high scalability; number of advertisements can be increased. Complexity of descriptions used is more complex in the IOPE algorithm due to SWRL annotations. Information retrieval guarantee of the IOPE algorithm is higher as shown in the experiments.

6.4. Threats to Validity

The matchmaker algorithm and the evaluation results presented in this research might be affected from several threats to validity, which need to be discussed.

One threat to validity is the complexity of the ontologies and the PE definitions used in the experiments. The size of the used ontology definitions and complexity of the PE definitions can be regarded as medium. To further generalize the obtained results, further studies on larger ontologies and more complex PE definitions are necessary. Running the matchmaker on different ontologies may reveal cases that are not handled in the algorithm.

Threats to evaluation results are mainly due to how precision and recall were measured. For the evaluation, precision and recall values were computed with respect to indications gave us by one judge, aware of the proposed matching algorithm approach. Although Voorhees, E. M. states that increasing the number of judges makes no improvement in comparing the performances of two systems, we will be more convinced if the experiments are also done with many judges [51].

Another threat to evaluation results can be due to the sample size (20 queries) that may limit the capability of statistical tests to reveal any effect. Increasing the number of queries during the experiments will increase the trustworthiness of the tests.

During the matchmaking of precondition and effect rules, we used a parameter based matching approach by decomposing the precondition and effect rules into their atom lists and then parsing the individual atoms (on which we begin the matching process). We did not consider inferencing from these rules to discover new information to be used during the matching process.

Matching and ranking involves the consideration of priorities. The scoring results of the algorithm are highly dependant on the coefficients that represent priorities. We prioritize output and effect similarity scores over input and precondition similarity scores. Another example is the WordNet score having a less priority than the ontology based similarity distance score. The coefficients used in the algorithm were chosen after many trials, but a machine learning procedure can be introduced to statically or dynamically calculate the best coefficient values.

7. Conclusions and Future Work

In this research, we have proposed and evaluated a novel approach for matchmaking condition expressions in OWL-S documents written in SWRL. In the matching, we use bipartite graph matching for finding the best pairings among arguments, predicates, and conditions belonging to advertisement and request. We also presented a detailed scoring calculation to be used as the weights of the bipartite graph. Scoring calculation includes subsumption, semantic distance, and WordNet components. The scoring based ranking capability enables us to include the Web services in the final result which would have been discarded in an inflexible classification. Lastly, we compared the IO and IOPE matching algorithms and showed that the information retrieval precision of the IOPE algorithm is significantly greater than the other.

At present, few academic models extend beyond simple subsumption based input and output matching. In terms of rule declarations, SWRL language is more often preferred to express the precondition and effect descriptions of services with respect to other allowed languages. The main contribution of this thesis to academic literature is twofold. Firstly, this research contributes a richer and more integrated service matching model. It is richer because it enables precondition an effect matching, and it utilizes various matching sub-modules in an integrated way. Secondly, with academic models predominantly focusing on general rule definitions for semantic Web services, our research adds a detailed analysis of the usage and decomposition of SWRL language for matchmaking needs to academic models.

From a practitioner's perspective, there is a need to study advanced semantic matching topics to provide wide usage of semantic Web applications in the industry. Our prototype semantic matching model can be a basis for semantic Web based industrial applications. We gave an example of sales transaction in this research where precondition and effect rules apply. If semantic Web services based e-commerce applications become prevalent, our model can be utilized while finding the services that a customer needs; in a

more filtered and clever way by taking into account the preconditions supplied by the customer and the effects desired.

We have developed a matchmaker for flexible Semantic Web service matchmaking and made a contribution to this area. However, the area of semantic matchmaking is broad and many aspects needs still to be investigated. We have concentrated on IOPE annotations in OWL-S. A number of issues were left out, but we are interested to examine them in the future.

Possible directions for future research consider improvements in matchmaker by involving new approaches in the Semantic Web area. In terms of PE matching we used a parameter based approach and didn't use a rule engine to infer new information. We may improve the matching process by improving the matching algorithm to include predicates, and may use a rule engine to infer new information. Some aspects such as security and privacy concerns can be included in the matching process. Here, much work can be done and various improvements can be achieved. 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. Non-functional attributes of Web services can also be taken into account in the matchmaking process. Finally, the implementation of the matchmaker is still in a prototype stage. Overall system performance and scalability aspects were so far only secondary goals. Here, further improvements are planned.

REFERENCES

1. Newcomer, E. and G. Lomow (2004). Understanding SOA with Web Services, Addison Wesley Professional.

2. Christensen, E., F. Curbera, et al. (2001). "Web Services Description Language (WSDL) 1.1." from http://www.w3.org/TR/wsdl.

3. XML Protocol Working Group. (2003). "SOAP Version 1.2 Part 0: Primer." From http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.

4. R. Fielding, J. Gettys, et al. (1999). "Hypertext Transfer Protocol -- HTTP/1.1." from http://www.w3.org/Protocols/rfc2616/rfc2616.html.

5. Miller, E. (2001). "Semantic Web Activity Statement." from http://www.w3.org/2001/sw/Activity.

6. Web Ontology Working Group. (2004). "OWL Web Ontology Language Overview." from http://www.w3.org/TR/owl-features/.

7. "Description Logics." from http://dl.kr.org/.

8. Martin, D., M. Burstein, et al. (2004). "OWL-S: Semantic Markup for Web Services." from http://www.w3.org/Submission/OWL-S.

9. Newton, G., J. Pollock, et al. (2004). "Semantic Web Rule Language (SWRL)." from http://www.w3.org/Submission/2004/03/.

10. "The Rule Markup Initiative." from http://www.ruleml.org.

11. Ilhan, E. S. (2007). Semantic Advanced Matchmaker (SAM). Computer Engineering. Istanbul, Bogazici University. Master: 91.

12. Bellwood, T., S. Capell, et al. (2004). "UDDI Version 3.0.2." from http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm.

13. "Universal Discovery Description and Integration Protocol (UDDI)." from http://uddi.xml.org/.

14. Daconta, M. C. (2003). "Designing the Smart-Data Enterprise." from http://web-services.gov/Designing the Smart-Data Enterprise.doc.

15. Schreiter, T. (2006). An Introduction into Semantic Web Services.

16. Akkiraju, R., J. Farrell, et al. (2005). "Web Service Semantics - WSDL-S." from http://www.w3.org/Submission/WSDL-S/.

17. Bruijn, J. d., C. Bussler, et al. (2005). "Web Service Modeling Ontology (WSMO)." from http://www.w3.org/Submission/WSMO/.

18. Klyne, G. and J. Carroll. (1998). "KIF- Knowledge Interchange Format." from http://logic.stanford.edu/kif/dpans.html.

19. Ghallab, M., A. Howe, et al. (1998). "PDDL - The Planning Domain Definition Language." from http://www.cs.yale.edu/pub/mcdermott/software/pddl.tar.gz.

20. Horrocks, I., P. F. Patel-Schneider, et al. (2003). "SWRL: A Semantic Web Rule Language Combining OWL and RuleML." from http://www.daml.org/2003/11/swrl/.

21. Marchiori, M. and S. Hawke. (2004). "Team Comment on the SWRL Submission." From http://www.w3.org/Submission/2004/03/Comment.

22. Antoniou, G. and F. v. Harmelen (2004). A Semantic Web Primer, MIT Press.

23. Baader, F., I. Horrocks, et al. (2005). Description Logics as Ontology Languages for the Semantic Web. Mechanizing Mathematical Reasoning, Springer Berlin / Heidelberg.

24. Beckett, D. (2004). "RDF/XML Syntax Specification." from http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

25. McBride, B. (2004). The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS. Handbook on Ontologies, Springer: 51-66.

26. Saip, H. A. B. and C. L. Lucchesi (1993). Matching Algorithms for Bipartite Graphs Technical Report, Universidade Estadual de Campinas.

27. Sirin, E., B. Parsia, et al. (2007). "Pellet: A Practical OWL-DL Reasoner." Web Semantics: Science, Services and Agents on the World Wide Web 5(2): 51-53.

28. Sirin, E. (2004). "OWL-S API." from http://www.mindswap.org/2004/owl-s/api/index.shtml.

29. Miller, G. A. (1995). "WordNet: a lexical database for English." Communications of the ACM 38(11): 39-41.

30. Pedersen, T., S. Patwardhan, et al. (2004). WordNet::Similarity - Measuring the Relatedness of Concepts. Proceedings of the Nineteenth National Conference on Artificial Intelligence San Jose, USA: 1024-1025.

31. Zar, J. H. (1998). Biostatistical Analysis, Prentice Hall.

32. Klusch, M., B. Fries, et al. (2005). OWLS-MX: Hybrid Semantic Web Service Retrieval. Proceedings of the 1st International AAAI Fall Symposium on Agents and the Semantic Web, Arlington VA, USA, AAAI Press.

33. Wang, Y. and E. Stroulia (2003). Semantic Structure Matching for Assessing Web-Service Similarity. Service-Oriented Computing - ICSOC 2003, Springer Berlin / Heidelberg. 2910/2003: 194-207.

34. Paolucci, M., T. Kawamura, et al. (2002). Semantic Matching of Web Services Capabilities. The Semantic Web — ISWC 2002, Springer Berlin / Heidelberg. 2342/2002.

35. Aversano, L., G. Canfora, et al. (2004). An Algorithm for Web Service Discovery through Their Composition (ICWS'04). Proceedings of the IEEE International Conference on Web Services. California, USA, IEEE Computer Society: 332-339.

36. Yao, Y., S. Su, et al. (2006). Service Matching Based on Semantic Descriptions. Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW'06) IEEE Computer Society: 126-126.

37. Dong-Wei, B., L. Chuan-Chang, et al. (2007). Web Services Matchmaking with Incremental Semantic Precision. Wireless Communications, Networking and Mobile Computing, IEEE Computer Society: 1-4.

38. Shen, X., X. Jin, et al. (2006). MSC: A Semantic Ranking for Hitting Results of Matchmaking of Services. Computer Software and Applications Conference (COMPSAC'06) IEEE Computer Society. 2: 291-296.

39. Guo, R., J. Le, et al. (2005). Capability Matching of Web Services Based on OWL-S. Database and Expert Systems Applications (DEXA'05) IEEE Computer Society: 653-657.

40. Wang, H. and Z. Li (2006). A Semantic Matchmaking Method of Web Services Based on SHOIN^+ (D)*. IEEE Asia-Pacific Conference on Services Computing (APSCC'06), IEEE Computer Society: 26-33.

41. ESSI WSMO working group. "WSMO." from http://www.wsmo.org/.

42. Li, L. and I. Horrocks (2003). A Software Framework For Matchmaking Based on Semantic Web Technology. Proceedings of the 12th international conference on World Wide Web. Budapest, Hungary, ACM: 331-339.

43. Ilhan, E. S. and A. B. Bener (2007). Improved Service Ranking and Scoring: Semantic Advanced Matchmaker (SAM). 2nd International Working Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'07), Barcelona, Spain.

44. Ilhan, E. S., G. B. Akkus, et al. (2007). SAM: Semantic Advanced Matchmaker. The Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'07). Boston, USA, Knowledge Systems Institute Graduate School: 698-703.

45. Senvar, M. and A. Bener (2006). Matchmaking of Semantic Web Services Using Semantic-Distance Information. Advances in Information Systems, Springer Berlin / Heidelberg. 4243/2006: 177-186.

46. Wu, J. and Z. Wu (2005). Similarity-based Web Service Matchmaking. IEEE International Conference on Services Computing (SCC'05), IEEE Computer Society. 1: 287-294.

47. Guo, R., D. Chen, et al. (2005). Matching Semantic Web Services across Heterogeneous Ontologies. Fifth International Conference on Computer and Information Technology (CIT'05) IEEE Computer Society.

48. Kuhn, H. W. (2005). "The Hungarian Method for the Assignment Problem." Naval Research Logistics 52(1): 7-21.

49. Khalid, M. A., B. Fries, et al. "OWL-S Service Retrieval Test Collection Version 2.1." from http://projects.semWebcentral.org/projects/owls-tc.

50. Küster, U., H. Lausen, et al. (2007). Evaluation of Semantic Service Discovery - A Survey and Directions for Future Research.2nd ECOWS Workshop on Emerging Web Services Technology (WEWST07).

51. Voorhees, E. M. (1998). Variations in Relevance Judgments and the Measurement of Retrieval Effectiveness. Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, ACM: 315-323.

52. Mann, H. B. and D. R. Whitney (1947). "On a Test of Whether One of Two Random Variables is Stochastically Larger Than the Other." the Annals of Mathematical 18(1): 50-60.

53. Rijsbergen, C. J. v. (1975). Information Retrieval, Butterworth.

54. XML Protocol Working Group. (2007). "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)." From http://www.w3.org/TR/soap12-part1/.

55. Wang, H., Z. Li, et al. (2006). An Unabridged Method Concerning Capability Matchmaking of Web Services. IEEE/WIC/ACM International Conference on Web Intelligence (WI'06), IEEE Computer Society: 662-665.

56. Kuang, L., J. Wu, et al. (2005). Exploring Semantic Technologies in Service Matchmaking. Proceedings of the Third European Conference on Web Services (ECOWS'05), IEEE Computer Society: 9.

57. Miller, E. (2001). "Semantic Web Activity Statement." From http://www.w3.org/2001/sw/Activity.

58. "Jena – A Semantic Web Framework for Java." from http://jena.sourceforge.net/.

59. McBride, B. (2002). "Jena: A Semantic Web Toolkit." IEEE Internet Computing 6(6): 55-59.

60. XML Working Groups. "Extensible Markup Language (XML)." from http://www.w3.org/XML/.

61. RDF Core Working Group. (2004). "RDF Vocabulary Description Language 1.0: RDF Schema." from http://www.w3.org/TR/rdf-schema.

62. "Web-Ontology (WebOnt) Working Group." from http://www.w3.org/2001/sw/WebOnt/.

63. W3C Working Group. (2004). "Web Services Architecture Requirements." From http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211/.

64. Berners-Lee, T., J. Hendler, et al. (2001). The Semantic Web: A New Form of Web Content that is Meaningful to Computers Will Unleash a Revolution of Possibilities. Scientific American Magazine.

65. Cardoso, J. and A. P. Sheth (2006). Semantic Annotations in Web Services. Semantic Web Services, Processes and Applications, Springer US. 3: 35-61.

66. Suwannopas, P. and T. Senivongse (2006). Discovering Semantic Web Services with Process Specifications. Distributed Applications and Interoperable Systems, Springer Berlin / Heidelberg. 4025/2006: 113-127.

67. Horrocks, I., P. F. Patel-Schneider, et al. (2004). "XML Concrete Syntax." From http://www.daml.org/2004/04/swrl/xmlsyntax.html.

68. Horrocks, I., P. F. Patel-Schneider, et al. (2004). "RDF Concrete Syntax." From http://www.daml.org/2004/04/swrl/rdfsyntax.html.

69. Luan, X., Y. Peng, et al. (2004). Quantitative Agent Service Matching. Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI'04) IEEE Computer Society: 334-340.

70. McBride, B. (2007). "An Introduction to RDF and the Jena RDF API." from http://jena.sourceforge.net/tutorial/RDF_API.

71. Maala, M. Z., A. Delteil, et al. (2007). A Conversion Process From Flickr Tags to RDF. Proceedings of the BIS 2007 Workshop on Social Aspects of the Web, Poznan, Poland.

72. Sánchez, D. and A. Moreno (2006). Discovering Non-taxonomic Relations from the Web. Intelligent Data Engineering and Automated Learning – IDEAL 2006, Springer Berlin / Heidelberg. 4224/2006: 629-636.

73. Salton, G. and M. J. McGill (1983). Introduction to Modern Information Retrieval, McGraw Hill.

74. TREC. (2006). "Common Evaluation Measures." from http://trec.nist.gov/pubs/trec15/appendices/CE.MEASURES06.pdf.

75. Buckley, C. and E. M. Voorhees (2000). Evaluating Evaluation Measure Stability. Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval. Athens, Greece, ACM: 33-40.

76. Dong, T., Q. Li, et al. (2007). An Extended Matching Method for Semantic Web Service in Collaboration Environment. 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD'07), IEEE Computer Society: 508-513.

77. Wilcoxon, F. (1945). "Individual Comparisons by Ranking Methods." Biometrics(6): 80-83.

78. Sycara, K., K. Decker, et al. (1997). Middle-Agents for the Internet. Proceedings of the 15th International Joint Conference on Artificial Intelligence. Nagoya, Japan.

79. Cross, V. (2004). Fuzzy Semantic Distance Measures Between Ontological Concepts. NAFISP'04, IEEE Computer Society. 2: 635-640.

80. Jang, J. e. a. (2005). Capability and Extension of a UDDI Framework for Semantic Enterprise Integration: UDDI Extension Framework for Incorporating Manufacturing Capability Profile. International Federation of Information Processing.

81. Giunchiglia, F., P. Shvaiko, et al. (2004). S-Match: An Algorithm and an Implementation of Semantic Matching. The Semantic Web: Research and Applications, Springer Berlin / Heidelberg. 3053/2004: 61-75.

82. Law, K. H. (2007). "Ontology: Basic Definitions and a Brief Introduction." from http://it.nees.org/support/workshops/2007/2wfcree/TN-2007-03_Law.pdf.

83. Ankolekar, A., D. Martin, et al. (2004). "OWL-S' Relationship to Selected Other Technologies." from http://www.w3.org/Submission/2004/SUBM-OWL-S-related-20041122.

84. Paolucci, M. and M. Wagner (2006). Grounding OWL-S in WSDL-S. IEEE International Conference on Web Services (ICWS'06): 913-914.

85. "Munkres' Assignment Algorithm." from http://216.249.163.93/bob.pilgrim/445/munkres.html.