FULLY AUTOMATIC ANNOTATION OF WEB SERVICE DESCRIPTIONS

(WEB SERVİS KOLEKSİYONLARININ TAM OTOMATİK ANLAMLANDIRILMASI)

by

Cihan AKSOY, B.S.

Thesis

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

in the

INSTITUTE OF SCIENCE AND ENGINEERING

of

GALATASARAY UNIVERSITY

May 2013

FULLY AUTOMATIC ANNOTATION OF WEB SERVICE DESCRIPTIONS

(WEB SERVİS KOLEKSİYONLARININ TAM OTOMATİK ANLAMLANDIRILMASI)

by

Cihan AKSOY, B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Date of Submission	: May 24, 2013
Date of Defense Examination	: June 18, 2013

Supervisors	: Dr. Vincent LABATUT
	Asst. Prof. Dr. Murat AKIN
Committee Members	: Prof. Bernard LEVRAT
	Assoc. Prof. Dr. Y. Esra ALBAYRAK
	Assoc. Prof. Dr. Temel ÖNCAN

ACKNOWLEDGEMENTS

First of all, I would like to present my sincere gratitude to Asst. Prof. Dr. Vincent Labatut, whose expertise, understanding, and patience, added considerably to my graduate experience. I appreciate his vast knowledge and skill in many areas, and his invaluable assistance in writing reports (i.e. tool documentations, publications and this thesis).

I would also like to thank my family for the support they provided me through my entire life and in particular, I must acknowledge my wife and my son, without whose love, encouragement and patience, I would not have finished this thesis.

In conclusion, I would like to thank all my teachers who guided me to improve myself with their great contributions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The definition and use of a semantic layer in Web Services (WS) descriptions is a prerequisite to the automation of several important operations such as WS composition and mining. For this reason, in the past years, many approaches have been proposed to either represent such high level information, or take advantage of it. In order to be properly tested and compared, these tools must be applied to an appropriate benchmark, taking the form of a collection of semantic WS descriptions. However, all of the existing publicly available collections are limited in terms of size or realism, leading to unreliable results. Large real-world syntactic (WSDL) collections exist, so an appropriate benchmark could be obtained through their semantic annotation. Due to the number of operations to process, performing this task manually would be costly in terms of time and efforts, though. A better solution would therefore be to use an automatic tool. With this motivation, in this work we propose a new fully automatic semantic annotation tool, called MATAWS, designed to process WS descriptions. The resulting tool takes advantage of the latent semantics present not only in the parameter names, but also in the data type names and structures. Concept-to-word association is performed by using Sigma, a mapping of WordNet to the SUMO ontology. The design of MATAWS is data-driven, based on the properties of ASSAM Full Dataset, the largest collection of real-world syntactic WS descriptions we could find. After having described in details our annotation method, we apply it to this collection and assess its efficiency. We also apply and evaluate MATAWS on another WS collection prepared by ourselves, in order to prove our tool performs well on new, independent data.

Keywords: Web Service, Semantic Web, Semantic Annotation, Ontology, WSDL, OWL-S, SUMO, WordNet.

RÉSUMÉ

La définition et utilisation d'une couche sémantique dans les descriptions des services Web (SW) est une condition préalable à l'automatisation de plusieurs opérations importantes telles que la composition et l'exploitation minière de SW. Pour cette raison, dans les dernières années, nombreuses approches ont été proposées pour soit représenter telle information de haut niveau, ou profiter de celui-ci. Afin d'être correctement testé et comparé, ces outils doivent être appliquées à un point de référence approprié, prenant la forme d'une collection des descriptions de SW sémantique. Cependant, toutes les collections existantes accessibles au public sont limitées en termes de taille ou de réalisme, ce qui conduit à des résultats peu fiables. Grandes collections syntaxiques (WSDL) du monde réel existent, donc un point de référence approprié peut être obtenu par leur annotation sémantique. En raison du nombre d'opérations à traiter, d'effectuer cette tâche manuellement serait coûteux en termes de temps et d'efforts, cependant. Une meilleure solution serait donc d'utiliser un outil automatique. Avec cette motivation, dans ce travail, nous proposons un nouvel outil entièrement automatique d'annotation sémantique, appelés MATAWS, conçu pour traiter les descriptions SW. L'outil fini profite de la sémantique latente présente non seulement dans les noms de paramètres, mais aussi dans les noms de type de données et des structures. Association de concept-à-mot est effectuée en utilisant Sigma, une façon de passage de WordNet à l'ontologie SUMO. La conception de MATAWS est guidée par les données, basée sur les propriétés de ASSAM Full Dataset, la plus grande collection des descriptions SW syntaxiques du monde réel que nous pourrions trouver. Après avoir décrit en détail notre méthode d'annotation, nous l'appliquons à cette collection et d'évaluer son efficacité. De plus, nous appliquons et évaluons MATAWS sur une autre collection de SW préparé par nous-mêmes, afin de prouver que notre outil marche bien sur nouvelles données indépendantes.

Mots clés : Service Web, Web sémantique, Annotation sémantique, Ontologies, WSDL, OWL-S, SUMO, WordNet.

ÖZET

Web Servis (WS) birleşimi ve madenciliği gibi alanlarda bir takım önemli işlevlerin otomatize edilebilmesi için WS tanımlarına eklenmek üzere anlamsal bir katmanın oluşturulması gereklidir. Bu nedenle, son yıllarda böylesine bir katmanı tanımlayan ya da bu tarz bir katmanı kullanan bir çok çalışma gerçekleştirilmiştir. Bu çalışmaların düzgün bir şekilde test edilebilmeleri için, ilgili çalışmalar, anlamsal WS tanımlarından oluşan, koleksiyon formundaki uygun bir kalite testine tabi tutulmalıdırlar. Ancak erişilebilir haldeki bu türden tüm koleksiyonlar, büyüklük ve gerçeklik bağlamında kısıtlıdırlar. Bu da güvenilebilir sonuçlar elde etmeyi engelleyici bir faktördür. Öte yandan, büyük ve halihazırda kullanılmakta olan, ancak anlamsal katman içermeyen koleksiyonlar (WSDL) da mevcuttur. Uygun bir kalite testi ortamı oluşturulması bu koleksiyonların anlamlandırılması ile de mümkün olabilir. Ancak koleksiyonların büyük olmasından ötürü yapılması gereken anlamlandırma işini elle gerçekleştirmek hem zaman hem de çaba yönünden çok maliyetli olmaktadır. Bu çalışmada, buradan yola çıkarak geliştirdiğimiz büyük WS koleksiyonlarının tam otomatik anlamlandırılmasını sağlayan MATAWS isimli aracımızı tanıtıyoruz. Aracımız WS tanımlarında potansiyel anlamlandırılabilecek olan parametre isimlerinden, veri tipi isimlerinden ve yapılardan faydalanmaktadır. Anlamlandırma, kelimelere hiyerarşik bir yapıdan kavram ilişkilendirme şeklinde olup, bu iş WordNet'ten SUMO ontolojisine eşlenmiş Sigma tarafından gerçekleştirilmektedir. MATAWS'ın tasarım süreci ASSAM Full Dataset isimli bir WS koleksiyonunun özellikleri baz alınarak ilerletilmiştir. Bu koleksiyon, araştırmalarımıza göre halihazırda kullanılan WS tanımlarından oluşan en büyük WS koleksiyonudur. Anlamlandırma yöntemimizi detaylı bir şekilde tanımladıktan sonra, yöntemi bu koleksiyona uyguladık ve verimliliğini değerlendirdik. Ayrıca MATAWS'ı halihazırda kullanılan WS tanımlarından oluşan ve kendimizin oluşturduğu bir başka koleksiyona daha uygulayıp sonuçları değerlendirdik, böylece aracımızın geliştirilirken baz aldığı koleksiyon haricinde, yeni ve bağımsız koleksiyonlar karşısındaki davranışını ve verimliliğini inceledik.

Anahtar Sözcükler: Web Servis, Semantik Web, Anlamlandırma, Ontoloji, WSDL, OWL-S, SUMO, WordNet.

1 INTRODUCTION

The semantic Web encompasses technologies which can make possible the generation of the kind of intelligent documents imagined ten years ago (Berners-Lee et al., 2001). It proposes to associate semantic metadata taking the form of concepts with Web resources. The goal is to give a formal representation of the meaning of these resources, in order to allow their automatic processing. The process of defining such associations is known as semantic annotation (or annotation for short), and generally relies on libraries of concepts collectively described and structured under the form of ontologies. The result is Web documents with machine interpretable mark-up that provide the source material for software agents to operate. The annotation of Web resources is obviously fundamental to the building of the semantic Web.

Besides static Web content such as textual or multimedia documents, semantic annotation also concerns dynamic content, and more particularly Web Services (WS). WS are non-static in nature; they allow carrying out some task with effects on the Web or the real-world, such as the purchase of a product. The semantic Web should enable users and agents to discover, use, compose, and monitor them automatically. As Web resources, classic WS descriptions such as WSDL files can be semantically enhanced using the annotation principle we previously described, i.e. by the association of various ontological concepts.

Efforts for WS annotation include WS semantic languages as well as tools to annotate legacy WSDL files. Several concurrent formats and technologies exist, leading to a profusion of research works. What interests us in this work, is the fact the ideas and tools resulting from these works must be tested. For this matter, one needs a large collection of semantic WS descriptions, in order to achieve statistical significance. Such collections exist, but are limited in terms of size, realism, and representativity.

Alternatively, the desired benchmark collection could be obtained by annotating a set of WSDL files. The annotation of WSDL files is much different from other Web resources, due to the specific structure of the document. One of the difficulties is the lack of context, since the available information is made up of isolated words, and not full sentences. A few publicly available tools exist for this purpose, as described in (Aksoy et al., 2011a). But those tools also have their own limitations, the main one being they are only partially automated and require human intervention, which is a problem when annotating a large collection of WS descriptions.

In this work we present MATAWS (Multimodal Automatic Tool for the Annotation of WS), a semantic WS annotator, whose purpose is to solve these limitations. MATAWS was designed with the objective of batch annotating a large collection of syntactic descriptions and generating a benchmark usable to test semantic-related approaches. It focuses on data semantics (i.e. the annotation of input and output parameters) contained in WSDL files, and currently outputs OWL-S files.

This work is not a single-handed project; it is based on former works conducted in our research group. Firstly, Nadin Kökciyan (Kökciyan, 2009) developed a tool able to extract networks from Web service descriptions in 2009. Next year, Yvan Rivierre (Rivierre, 2010) improved this extractor with new features while Koray Mançuhan and me, we started to develop the first version of MATAWS as our BSc thesis (Aksoy & Mancuhan, 2010). In our tool, we benefited from the tool developed by Nadin and Yvan in order to represent the WS collections as Java objects and to work on them. Our goal was the same: to annotate the objects that represents WS collections by developing methods that reduce annotation cost. In my master thesis, I extended this work by improving and extending MATAWS in various ways, as described in this document.

The second section of this thesis is a state of the art of WS. The third section describes the tool we have developed. The fourth section presents and compares the results obtained after having applied our tool on two different collections. It includes quantitative and qualitative analysis of the results. Finally, in the conclusion we summarize our work, discussing its limitations and possible improvements that would address them. Also we mention organizational challenges we encountered during the project and how we have resolved them.

2 STATE OF THE ART

This section aims at giving the reader the prerequisites needed to understand the description of our tool. First, we introduce the notions of WS and semantic descriptions, with their related standards. We highlight the need of an appropriate collection of semantic description to test modern tools. We therefore make a review of the existing publicly available semantic WS collections. It turns out none of them satisfies our criteria. So, we propose to constitute a semantic collection by annotating an appropriate syntactic one. For this purpose, we review the existing publicly available syntactic WS, and the existing annotation tools. We highlight the limitations of the latter, which conducted us to design our own annotation tool, described in the next section.

2.1 Web Services

A Web service (WS) is a software component able to perform a set of well-defined tasks, and it can be remotely invoked through a stack of standard technologies (Cabral et al., 2004).

It can be implemented in any language, deployed on any platform, because it is wrapped in a layer derived from XML standard, which makes its invocation platform-independent. It can be dynamically discovered and invoked by other WS. For example, when we want to make a trip, it is enough to interact with one WS; it will fix all we need, such as hotel reservation, purchasing airline tickets, rental car, etc. (see Figure 2.1).

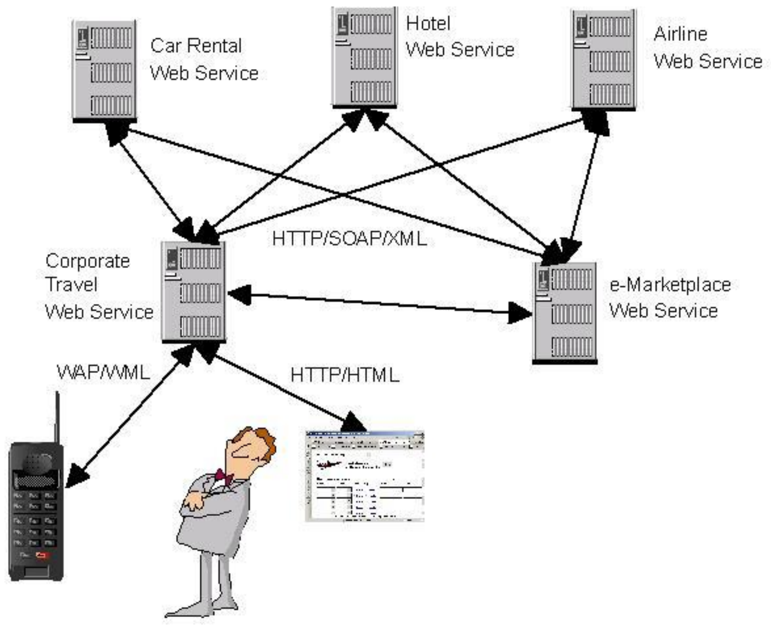


Figure 2.1 Business travel application interacts with WS (Ryman, 2003).

This technology, initiated by Microsoft and IBM, then partially standardized under the auspices of the W3C, is now accepted by all players in the IT industry without exception (Budinoski et al., 2010).

2.1.1 Terms of Reference

The concept of WS is currently organized around three technologies (Christensen et al., 2001; Clement et al., 2005; Gudgin et al., 2007):

- **SOAP** (Simple Object Access Protocol) is a platform independent exchange protocol inter-applications based on XML. SOAP call is a stream in ASCII XML tags and transported with HTTP protocol.
- **WSDL** (Web Service Definition Language) describes WS in the XML format specifying the methods that can be invoked, their signature and the access point (URL, port, etc.).
- **UDDI** (Universal Description, Discovery and Integration) standardizes distributed directory solution for WS, allowing both the publication and exploration. UDDI behaves itself as a WS whose methods are called via SOAP.

These standards will be detailed in the section 2.1.3.

2.1.2 Reference Architecture

The WS model is based on three main units as seen in Figure 2.2:

- **Service provider**: It creates the WS and possibly publishes its interface and access information. The service provider must determine the WS will be exposed, the tradeoffs between security and easy availability and price of exposed WS.
- **Service requester**: It demands the published WS by the provider. There is information about how it will invoke the WS and the parameters for service registry.
- **Services registry**: It takes WS descriptions which are published by providers and allows them to search and find these descriptions.

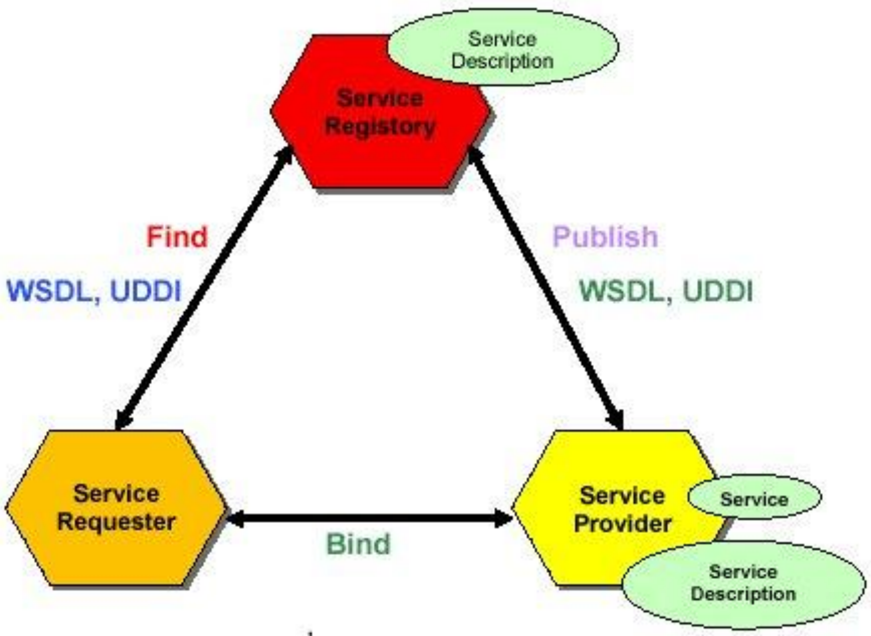


Figure 2.2 WS Model (Kreger, 2001).

There are six steps for invoking a WS of provider by a WS requester (see Figure 2.3):

1. The WS requester searches and finds WS that it wants in the service registry.
2. The requester prepares a SOAP message based on XML.
3. The requester sends the message to the SOAP request listener found in the web server or application server.
4. The SOAP server analyzes the request and invokes the necessary object with parameters.
5. The result of invoked method is returned to the SOAP server. The server converts the result as a SOAP message and returns it to the requester.
6. The requester sends the result to the program by extracting the necessary information from SOAP message.

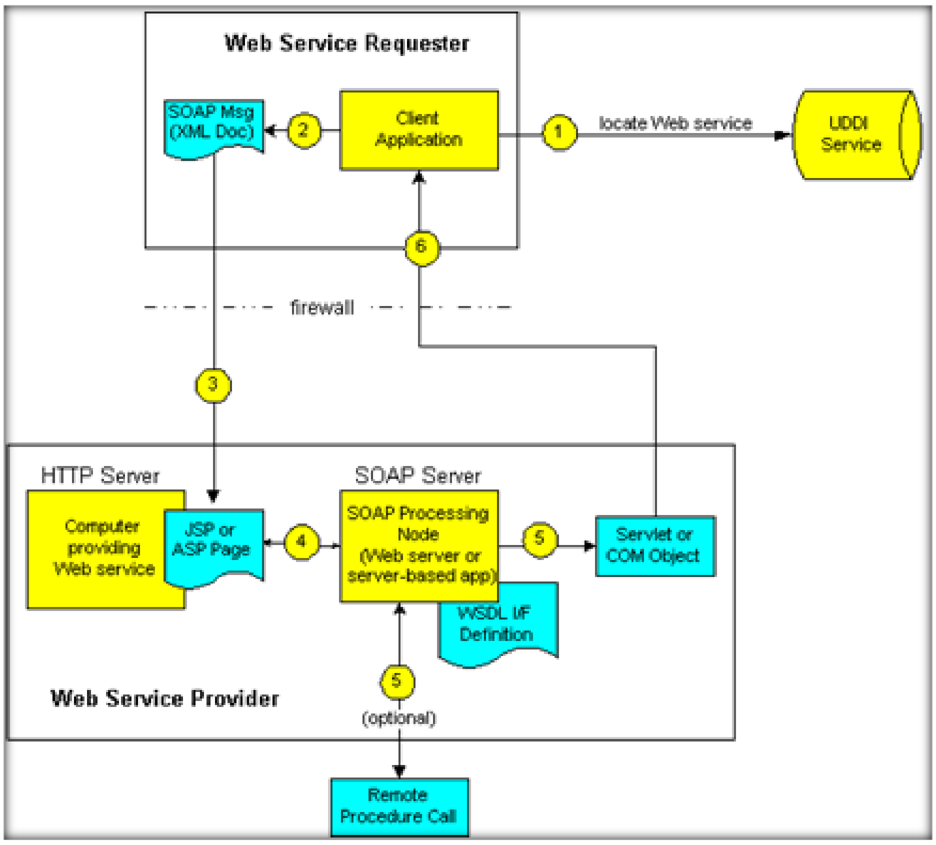


Figure 2.3 Main steps between the requester and the provider of WS (Kreger, 2001).

2.1.3 Web Service Standards

In this section, we describe the three main standards of WS: SOAP, WSDL and UDDI. Our explanations do not mean to be exhaustive: we focus only on the points relevant with our needs.

2.1.3.1 SOAP

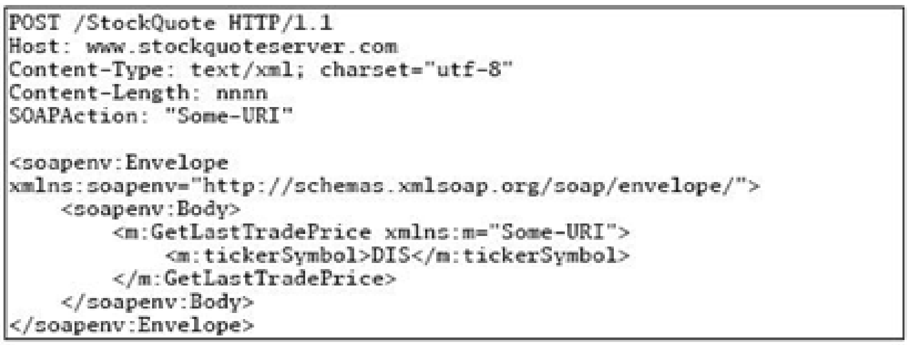
SOAP is a communication specification between WS by exchanging XML messages over the Web. SOAP is simple and easy to implement in web servers or application servers. It is independent of programming languages and operating systems used for the implementation of the WS. SOAP designers have indeed succeeded to preserve the most important generality in the XML representation.

SOAP is defined as a lightweight protocol for exchanging data in a peer to peer network, that is to say decentralized. Based on XML, SOAP provides a simple mechanism for representing different aspects of a message between applications. Since it does not impose any specific programming model, SOAP can be used in all styles of communication: synchronous or asynchronous, point to point or multipoint, intranet or Internet.

The SOAP specification is divided into four parts (Chauvet, 2002):

- SOAP envelope, that defines the context of a message, its destination, its content and options.
- SOAP encoding rules, defining the representation of application data in the body of a SOAP message (particularly the structure).
- An RPC protocol, defining the sequence of requests and responses.
- The definition of the use of HTTP as a transport layer of SOAP messages.

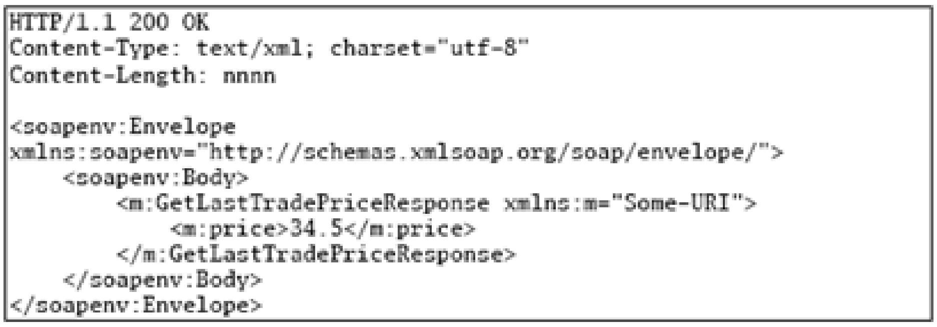
Coding rules make extensively use XML Schema to describe the constitutive data structure of the SOAP messages. On the Figure 2.4 and Figure 2.5 the request and response of the SOAP message are seen.



```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
            <m:tickerSymbol>DIS</m:tickerSymbol>
        </m:GetLastTradePrice>
    </soapenv:Body>
</soapenv:Envelope>
```

Figure 2.4 Example of SOAP request (Ryman, 2003).



```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <m:GetLastTradePriceResponse xmlns:m="Some-URI">
            <m:price>34.5</m:price>
        </m:GetLastTradePriceResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Figure 2.5 Example of SOAP response (Ryman, 2003).

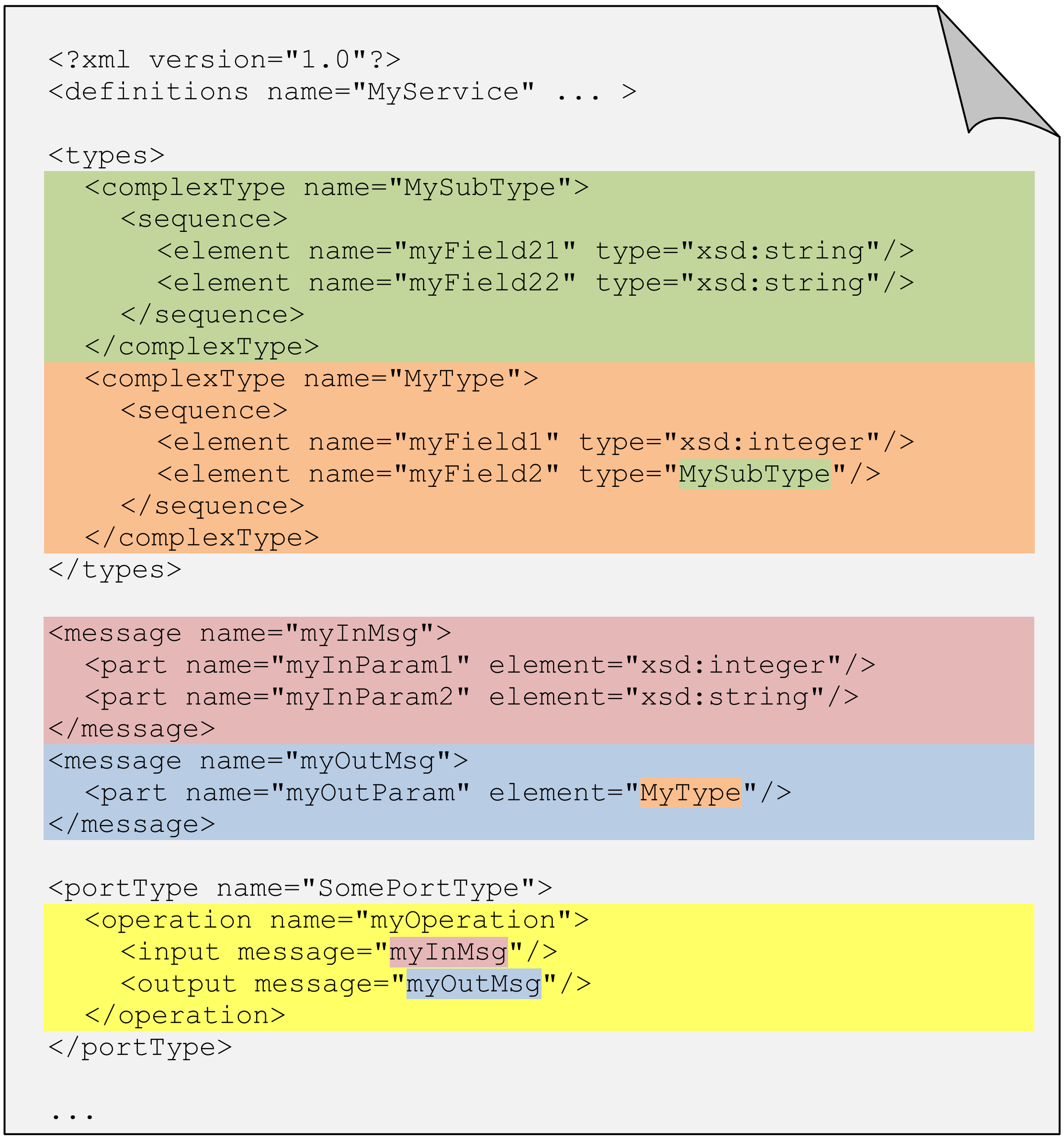
2.1.3.2 WSDL

The *de facto* standard is WSDL (WS Description Language), an XML-based language specified in 2001 (Christensen et al., 2001) by the World Wide Web Consortium (W3C). It allows defining the public interface of WS, and more particularly: the communication protocol to be used, the format of the messages to be exchanged, the methods the client can invoke, and the location of the service. A second version was approved in 2007 to become a W3C recommendation. However it is much less used, so even if it contains significant changes, we chose to focus on version 1.1.

In this work, we are more particularly interested in the higher-level elements of WSDL, i.e. those related to the programmatic use of the service. From this perspective, WSDL files are very similar to RPC or RMI interfaces. They describe a list of *operations*, corresponding to the functions/methods the client can remotely invoke. The inputs and outputs ossf an operation are described under the form of two so-called *messages*: one for the inputs, the other for the outputs. A message is a group of *parts*, which correspond roughly to parameters, in terms of programming. Each one is characterized by a *name* and a *data type*. Figure 2.6 gives an example of simple WSDL file. It contains a single operation `myOperation` (highlighted in yellow) whose input and output messages are `myInMsg` and `myOutMsg`, respectively. The former (in red) contains two parts `myInParam1` and `myInParam2`, whereas the latter (in blue) has only one, named `myOutParam`.

Of course, all these elements are described in a neutral, implementation-independent way. For this matter, the data types are defined using the *XML Schema Definition* language (XSD), a 2001 W3C recommendation initially designed to let users define their own XML grammars (Fallside & Walmsley, 2004). WSDL actually uses only the part of XSD related to the definition of data types (Biron & Malhotra, 2004). Note WSDL theoretically allows using other type definition languages than XSD, but as far as we know only this one has been used in production WS, up to now.

XSD types are generally separated in two groups: simple- and complex-content types. The former correspond to XML elements containing directly a value, and nothing else: no attribute, no other element. From the programming point of view, they can be implemented using the predefined simple types present in most languages: integer, float, string, etc., or derived types such as enumerations. The latter point out at elements containing at least one attribute and/or one element. Their implementation requires defining custom classes, for object-oriented languages, or types such as union, structures, arrays, etc. for non-object languages.



```
<?xml version="1.0"?>
<definitions name="MyService" ... >

<types>
  <complexType name="MySubType">
    <sequence>
      <element name="myField21" type="xsd:string"/>
      <element name="myField22" type="xsd:string"/>
    </sequence>
  </complexType>
  <complexType name="MyType">
    <sequence>
      <element name="myField1" type="xsd:integer"/>
      <element name="myField2" type="MySubType"/>
    </sequence>
  </complexType>
</types>

<message name="myInMsg">
  <part name="myInParam1" element="xsd:integer"/>
  <part name="myInParam2" element="xsd:string"/>
</message>
<message name="myOutMsg">
  <part name="myOutParam" element="MyType"/>
</message>

<portType name="SomePortType">
  <operation name="myOperation">
    <input message="myInMsg"/>
    <output message="myOutMsg"/>
  </operation>
</portType>

...
```

Figure 2.6 Excerpt of simple WSDL file (non-relevant parts have been removed).

XSD is powerful in the sense it allows defining many different sorts of types, especially for complex contents. However, between the constraints imposed by WSDL and the choices made by WS designers, only two kinds of data types turn out to be used in practice: simple-content types, and *sequence* complex-content types. For this reason, in the rest of this article we focus on these sorts of types. *Sequence types* lead to elements containing other elements of specific types and in a predefined order. They can be implemented as classes or structures, the internal elements corresponding to fields. The classes/structures can even be recursive, since each contained element can itself have a sequence type. In Figure 2.6, the input parameters `myInParam1` and `myInParam2` both have a predefined simple type: `integer` and `string`, respectively. The output parameter `myOutParam` has a custom type `MyType`, highlighted in orange. It is a sequence of two elements: an `integer`, and another element of type `MySubType` (in green) which contains itself two `string` values. In the rest of the article, when we mention complex-content types, we implicitly refer to those based on sequences.

2.1.3.3 UDDI

One must know the providers of WS and provided WS to be able to use a WS. UDDI allows providers to publish their WS by describing them, and it provides requesters to scan and find published WS. UDDI Business Registries are servers that hold information of companies and their WS. These servers store information in the database that come from service providers, and allows the others accessing this information. There are two business registers currently, one of them is uddi.microsoft.com and the other is uddi.ibm.com. These servers share obtained information with other servers to increase the access speed as seen on Figure 2.7. UDDI servers realize recordings, modifications and scannings using WS (with SOAP messages) (OASIS, 2004).

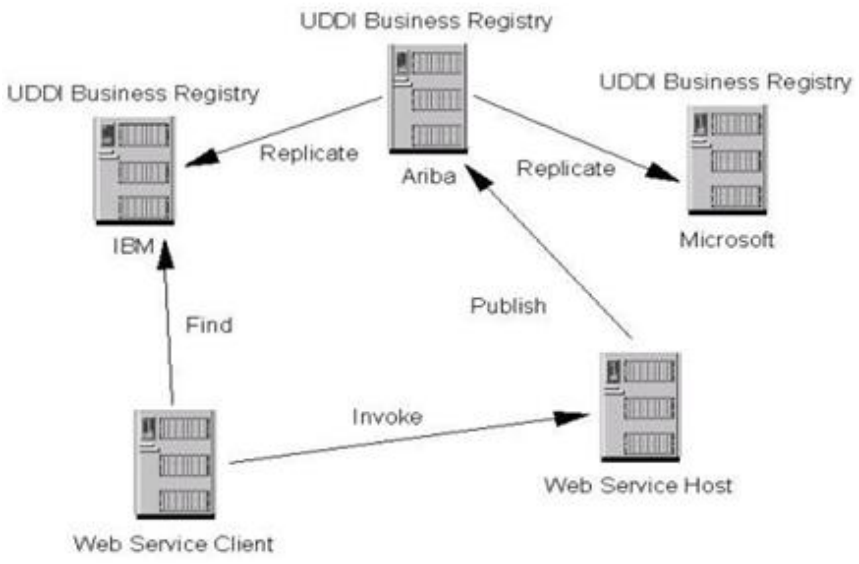


Figure 2.7 Business Registry Servers (OASIS, 2004).

2.2 Semantic Layer

The amount of information available on the internet today is huge and it is growing exponentially, the number of internet users is doubling every year. We believe that the size of the Web covered by the search engines is estimated at least 20 billion pages. But the specificity of such information sources makes them difficult to use. The main reason is that the documents are fragmented, dispersed, heterogeneous and often unstructured. However, thanks to the efforts of the semantic Web community (W3C), a second generation is established with a vision initiated in 1998 by Sir Tim Berners-Lee (Berners-Lee, 1998) aiming to structure available information on the Web. In this way, it is possible to load the contents of the meanings and allow machines to carry out automated tasks.

The semantic Web is a semantic layer added over several existing technologies as seen on Figure 2.8. The semantic Web has all the features of the existing Web. Processed semantic information is not expressed in natural language anymore, but formalized to be automatically processed.

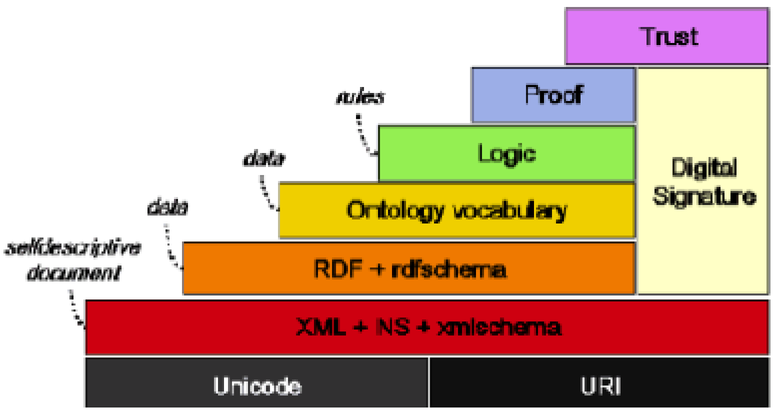


Figure 2.8 Layers of semantic Web (Hyvönen, 2002).

A central aspect of the infrastructure is its ability to identify and locate various resources. It is based on the notion of URI (Uniform Resource Identifier) which allows assigning a unique identifier to a set of resources on the Web but also in other areas (documents, mobile phones, people, etc.). It is also at the base of the W3C languages.

Another feature of these languages is to be systematically expressible and exchangeable in an XML syntax. This allows taking the advantage of all the technologies developed around XML: XML Schemas, XML resource exploitation tools (Java libraries, etc.), databases that manage XML files, although specific query languages are necessary for languages built on XML as RDF.

2.2.1 Terms of Reference

The main technologies in the semantic WS area (Brickley & Guha, 2004; Klyne & Carroll, 2004; McGuinness & Harmelen, 2004) are:

- **RDF (**Resource Description Framework): This is a conceptual model that can describe any data in triple format.
- **RDF-S** (RDF Schema): This is the language that is used to create vocabularies to describe things.
- **OWL** (Web Ontology Language): This is an XML dialect based on RDF. It enables defining structured Web ontologies, in other words, it allows you to define terminologies to describe concrete domains. Terminology is made up of concepts and properties. A domain consists of instance concepts.

2.2.2 Semantic Web Services (SWS)

SWS is at the convergence of the semantic Web and WS. Even if the notion of WS allows more benefiting from internet technology, it cannot be automated. This means WS cannot be properly discovered and composed due to lack of certain knowledge. The goal of the SWS concept is to create a semantic Web of WS whose properties, capabilities, interfaces and effects are described unambiguously and exploitable by machines. Therefore, semantics expressed will allow automating the following features, which are necessary for effective business collaboration:

- WS description and publication process,
- WS discovery,
- WS selection,
- WS composition,
- WS administration.

2.2.3 Ontology

In computer and information sciences, an ontology is a structured set of terms and concepts representing the meaning of an information field, whether as a metadata namespace, or as the elements of a certain domain. The ontology is a data model representing a set of concepts but also the relationships between these concepts. It is used to make inference on the objects of a domain (Gruber, 1993).

We distinguish different levels of ontologies according to the modeled domain and the tasks for which they are designed:

- The domain ontologies are specific to a domain and have a good precision.
- The general ontologies are not specific to a domain. Their precision is average.
- The superior ontologies (upper level ontologies) represent general concepts like space, time or matter. They are universal.

There are many ontologies created according to the needs like ontologies related to biomedical, tourism, finance, etc[1].

2.2.4 Extended Architecture

The extended architecture is the reference architecture embellished with a layer of semantics. Figure 2.9 shows an example of such an architecture. The stack consists of several layers, each layer based on a particular standard. Above the transport layer, there are three layers forming the basic infrastructure described previously. Security

[1] http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library

layer is responsible for the security of all layers. Transactions layer handles transactional interoperability of WS. Administration layer is responsible for the administration of WS. QoS (Quality of Service) layer is responsible for contracts between a service provider and a service consumer (Kellert & Toumani, 2004).

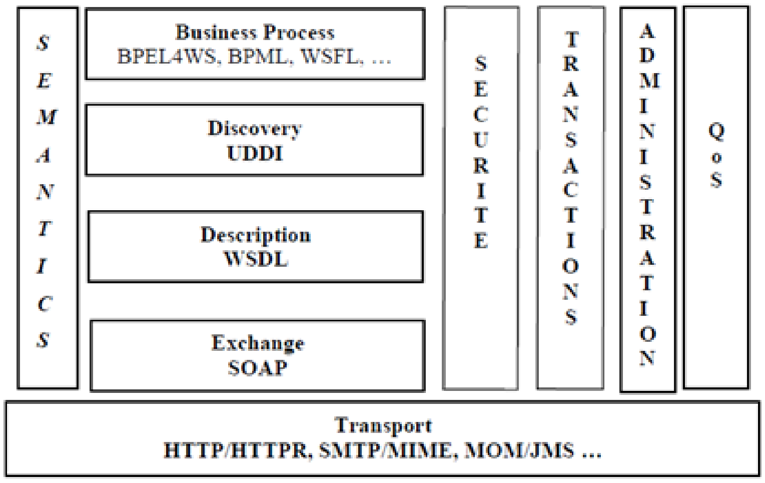


Figure 2.9 WS Stack (Kellert & Toumani, 2004).

Several languages were defined to represent semantic descriptions, the most prominent being OWL-S (Martin et al., 2004), WSMO (De Bruijn et al., 2005), WSDL-S (Akkiraju et al., 2005) and SAWSDL (Farrell & Lausen, 2007). As our goal in this study is to provide the community a collection of WS descriptions usable as a benchmark, we decided to use OWL-S as the output language of MATAWS, since it is the most widespread in the literature, and it is supported by the W3C. That’s why we only present OWL-S in the following section.

2.2.5 OWL-S

The OWL-S format meets all the criteria for semantically enriching WSDL descriptions, with the help of OWL ontologies and the semantic Web. The OWL-S approach has its origins in the fields of artificial intelligence where it was used to describe the capabilities of agents in complex and heterogeneous systems. OWL-S intelligently combines the expressiveness of the algorithms descriptions, DL (Descriptions Logic), with the pragmatism that we find in the current standards (SOAP, WSDL), to describe the WS in a semantic way, but also to aggregate them by respecting formalized data types (Cabral et al., 2004).

OWL-S consists of three high-level ontologies (see Figure 2.10): *Service Profile*, *Service Model* and *Service Grounding*. The first semantically describes the functionalities of a WS with additional options to help interpretation. The second provides a specific and detailed description of the sequence of actions that the provider will follow when interacting with a consumer. It aims to give some autonomy to WS consumers who can easily deduce interaction protocol to be used and the concrete consequences of each exchange message. The third is a detailed semantic description about mapping of abstract information of exchanges to real standard messages that the provider and the consumer exchange (Paolucci et al., 2003). These modules are more detailed in the following subsections.

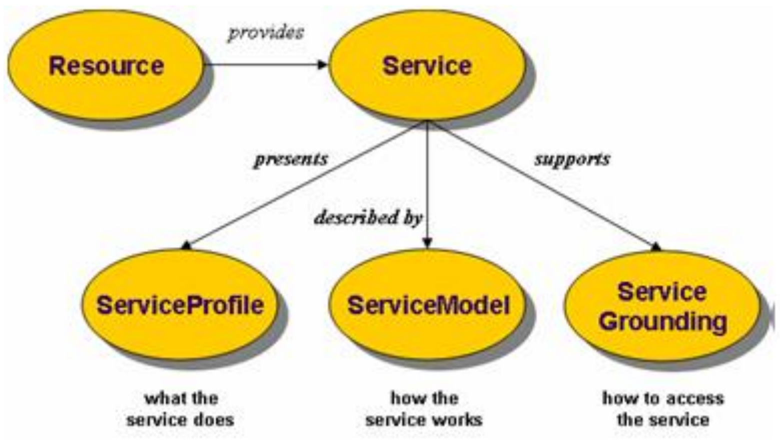


Figure 2.10 OWL-S Module Organization (Paolucci et al., 2003).

2.2.5.1 Service Profile

An OWL-S Profile aims to describe the functionalities of a WS. OWL-S allows describing these functionalities as performing a transformation process. This transformation takes place at two levels: at the data level, a set of inputs are transformed into a set of outputs; at the more concrete level, a set of conditions are averred while others are invalidated. If we take an example to understand this point, we can consider a travel booking WS: at the data level, it waits a departure and arrival as input, at the concrete level it reserves the flight, generates a ticket, and debits the consumer's credit card.

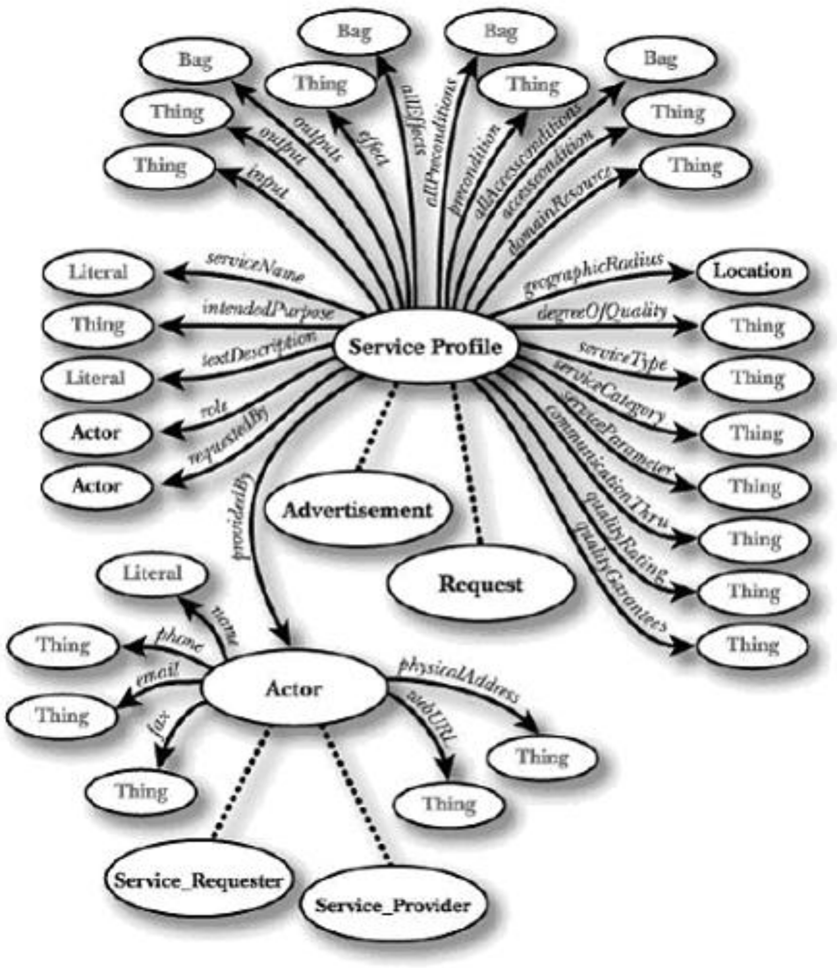


Figure 2.11 OWL-S Profile Ontologies (Paolucci et al., 2003).

Figure 2.11 illustrates the ontology of OWL-S Profiles. This figure is logically divided into three parts:

- The lower part is defining the Actor. It represents the type of vendor-specific information.
- In the center, the functional attributes description is found. From the estimate *Quality Rating* of the WS, to the *Geographic Radius* that specifies whether spatial constraints applying to WS or not. As an example, such a constraint would prevent a need for Chinese food from London is served in a restaurant in Shanghai.
- The upper part of the figure shows the functional description of the WS. This means the capacity of WS in terms of inputs, outputs, preconditions and effects. An input required by a WS to produce an output. Pre-conditions are the conditions that must be averred in the real world for the proper execution of the WS. The implementation of the WS results in actions in the real world; these are described as effects of the agent.

Considering the example of the WS flight reservation, the inputs are departure and arrival of the desired flight. The output is the confirmation that the order has been received and successfully processed. A precondition would be a valid credit card. As effect, the credit card is debited and the flight is assigned.

2.2.5.2 Service Model

The Service Model performs two tasks; the first is to specify the interaction protocol allowing the consumer to know which information is expected and which information will be sent back by the provider at a specific time of the transaction. As the second, it allows the consumer to know what the provider does with this information in a way provider makes public the specific treatments.

The Service Model is used to express any kind of flow control structure, including loops, sequences, conditions, non-determinant choices and competition. The processes are defined as transformations between an initial state and a final state. The final state is defined by the process inputs and a set of pre-conditions for a process without error. Process result is defined as a set of outputs and a set of effects in the real world. OWL-S distinguishes between two types of inputs and outputs: the first type is internal inputs and outputs, i.e. a process output feeds a process input of a next chain execution. The latter type is called external inputs and outputs: they define data provided by the consumer, which will be transferred to a provider.

During the interaction with the provider, the consumer analyzes the Service Model to infer which process is being done by the provider. The consumer is particularly interested with the inputs that provider needs and outputs resulting from the execution of this treatment. Indeed, most of the time the consumer will be responsible for providing inputs and interpret the outputs. Following the Service Model, and interpreting the information collected by the provider, the consumer can know at what time it should send the next information. Implicitly, the Service Model of the provider also specifies the interaction protocol between the provider and the consumer.

2.2.5.3 Service Grounding

The Service Grounding transforms the abstract description of the information exchanged between the provider and the consumer to the messages that can be exchanged asynchronously, or via a remote procedure call. Specifically, Service Grounding is defined as a 1-1 mapping between atomic processes and WSDL specifications of messages. It takes the WSDL definition and links of abstract messages, whereas the information that is used for composing messages is extracted through the execution of the Service Model. The integration of WSDL in OWL-S descriptions facilitates interaction between WS without OWL-S and WS with OWL-S to describe their functionalities.

2.3 Benchmarking Semantic Web Services Tools

As we mentioned in the introduction, the semantic WS area is highly dynamic, and many searchers produce tools related to semantic WS. Those should be tested on an appropriate semantic WS collection so that they can be properly evaluated. Then, it is necessary first to identify an appropriate semantic WS collection, i.e. one representative enough. For this purpose, we define two criteria: first the description files need to represent real-world services in order to contain realistic data, and second the collection must be large so that it is general enough.

In this section, we first review the existing semantic collections. This leads us to the conclusion there is no satisfying semantic WS collection. We then consider an alternative, which consists in generating one by annotating an existing syntactic WS collection. For this purpose, we first review the existing syntactic collections and annotation tools. We highlight the limitations of the latter, which lead us to develop our own tool.

2.3.1 Semantic Collections

The main publicly available collections of semantic WS are those provided by the ASSAM WSDL Annotator project, SemWebCentral and OPOSSum. Their major features are gathered in Table 2.1.

The ASSAM WSDL Annotator project (Automated Semantic Service Annotation with Machine learning) (Hess, 2004) includes two collections of WS descriptions named *Full Dataset* and *Dataset2*. *Full Dataset* is a collection of categorized WSDL files, which contains 816 WSDL files describing real-world WS. *Dataset2* is a collection of OWL-S files, obtained by annotating a subset of the WSDL files using the ASSAM Annotator (cf. section 2.3.3). 164 descriptions were fully labeled, assigning ontology references to the WS itself, its operations and their inputs and outputs.

Table 2.1 Collections of semantic WS descriptions: main features.

Name	Dataset2	OWLS-TC4	SAWSDL-TC	SWS-TC
Source	ASSAM project	SemWeb Central	SemWeb Central	SemWeb Central
Type	Real-world descriptions	Real-world descriptions, partially resampled	Real-world descriptions, partially resampled	N/A
Language	OWL-S	OWL-S	SAWSDL	OWL-S
Annotated Type	Data, Functional	Data	Data	Data
Size	164	1083	894	241
Particular features	Processed using Assam annotator	Single interface, one operation per service	Single interface, one operation per service	N/A

SemWebCentral[2] is a community whose purpose is to gather efforts from people working in the semantic Web area. Three semantic collections are available: *OWLS-TC* (OWL-S Test Collection), *SAWSDL-TC* (SAWSDL Test Collection) and *SWS-TC* (Semantic WS Test Collection). OWLS-TC4 is the fourth version of this test collection. It provides 1083 semantic descriptions written in OWL-S from seven different domains. Part of the descriptions were retrieved from public IBM UDDI registries, and semi-automatically transformed from WSDL to OWL-S. SAWSDL-TC originates in the OWLS-TC collection. It was subsequently resampled to increase its size, and converted to SAWSDL. The collection provides 894 semantic WS descriptions. The descriptions are distributed over the same seven thematic domains than OWLS-TC. SWS-TC is a collection of 241 OWL-S descriptions. There is not much information about this collection.

OPOSSum (Online POrtal for Semantic Services) (Küster et al., 2008) is a joint community initiative for developing a large collection of real-world WS with semantic descriptions. Its aim is to create a suitable test bed for semantically enabled WS technologies. OPOSSum gathered the three semantic collections of SemWebCentral, plus the *Jena Geography Dataset* collection, explicitly collected within OPOSSum.

[2] http://wwwprojects.semwebcentral.org/

The collection contains 201 real-world WS descriptions retrieved from public. All the described WS belong to the domains of geography and geocoding. Unfortunately, for now, no semantic descriptions are available for the services of the Jena Geography Dataset, which is why this collection is absent from Table 2.1.

These collections have been widely used in semantic WS-related works (Ma et al., 2008; Skoutas et al., 2008). As shown in Table 2.1, they all focus on the annotation of the data elements, which corresponds to our objective. However, one can notice some limitations. SWS-TC description is insufficient, it is not even clear if the WS descriptions are real-world. Dataset2 contains only real-world WS descriptions but it is very small, which can raise questions about its representativity. On the contrary, OWLS-TC4 and SAWSDL-TC contain a substantial number of descriptions. Nevertheless, these have been partially resampled in an undocumented way, which raises important questions concerning their realism.

2.3.2 Syntactic Collections

From the previous section, we can conclude there is no appropriate public collection of semantic WS descriptions since they don't respect one or both representativeness criteria. One way of obtaining such a dataset consists in annotating a collection of syntactic descriptions. But for this purpose, it is necessary first to identify an appropriate syntactic collection. In this section, we review and compare those publicly available.

There are not a lot of WSDL collections publicly available, so it is possible to review them all here. The *SemWebCentral* website centralizes various resources related to the semantic Web. It offers several collections of semantic WS descriptions using various semantic formats. Amongst them, the *OWLS-TC4* (OWL-S Service Retrieval Test Collection v.4) collection provides both WSDL and OWL-S descriptions for 1083 services. However, only a part of the descriptions concerns real-world WS, and those are not identified in the collection. Moreover, the way the rest of the WSDL files were obtained is not documented.

The *OPPOSum* (Online Portal for Semantic Services) website gathers various collections related to both semantic and syntactic WS descriptions (Küster et al., 2008) (including some of SemWebCentral). For our purposes, the most noticeable one is the *Jena Geography Dataset*, which contains 201 real-world descriptions from the domain of geography and geocoding. The size of this collection is therefore far away from representativity.

Several IEEE conferences included some challenges based on the processing of syntactic and/or semantic WS descriptions. This is the case of the 2005 IEEE International Conference on e-Business Engineering (ICEBE)[3], and also of the joint IEEE Conference on Commerce and Enterprise Computing (CEC) and IEEE EE Conference on Enterprise Computing, E-Commerce and E-Services (EEE) between 2005 and 2010[4]. During these competitions, the evaluation was performed using some benchmarks specially defined for the occasion, available on the websites of these conferences. These collections are very large. However, they were all created artificially, which makes them inappropriate to the representativity matters.

Table 2.2 Collections of WS descriptions: main features.

Name	OWLS-TC4	Jena Geography Dataset	Discovery & Composition Challenge	Testsets	Full Dataset
Source	SemWeb Central	OPPOSum	ICEBE	CEC & EEE	ASSAM Project
Type	Real-world descriptions, partially resampled	Real-world descriptions	Fully-sampled	Fully-sampled	Real-world descriptions
Size	1083	201	23409 & 26904	From 351 to 5170	816

[3] http://www.comp.hkbu.edu.hk/~ctr/wschallenge/
[4] http://cec2008.cs.georgetown.edu/wsc08/

The *ASSAM project* (Automated Semantic Service Annotation with Machine learning) (Hess et al., 2004) includes a collection of WSDL files named *Full Dataset*. It gathers 816 real-world WS descriptions, containing a 7877 operations in total. As seen on the Table 2.2, it is the largest collection made up of real-world descriptions available up to now, which supports well the representativity concerns.

2.3.3 Annotation Tools

Since we could identify an appropriate syntactic collection, it is now necessary to annotate it, in order to get the benchmark, i.e. the semantic collection. In general, annotating consists in tagging the considered resource with labels whose semantic is precisely known. Although these tags can take various forms such as terms from a controlled vocabulary, ontological concepts are the preferred approach for WS. The relational information encoded in the ontology allows building a hierarchical structure of concepts, which in turns can be used to perform automatic inference.

The annotation process, which consists in tagging the considered resource with the appropriate concepts, is completely dependent on both the nature of the resource and the automation goals. In other words, one will not annotate a text the same way one annotates a picture. And for a given kind of resource, the relevance of its various components might vary depending on the process one wants to automate. As shown in section 2.1.3.2, WS descriptions are much different from other Web resources in terms of content. Moreover, their use also significantly differs: their semantic description aims at automating WS discovery, invocation, composition and monitoring (Martin et al., 2004). Amongst the many elements constituting a WS description, these specific properties led to the identification of four thematic groups (Sheth, 2003): those concerning the inputs and outputs of the WS (*data* semantics), the process performed by the WS (*function* semantics), the constraints applied to the WS, such as the quality of service (*non-function* semantics) and the execution flow inside the WS (*execution* semantics).

Although semantic descriptions allow annotating all parts of the description, the focus in the literature is on the data semantics, and more specifically the exchanged parameters. This is certainly due to the fact this information is particularly important to automate WS discovery and composition, two popular research fields. As mentioned before, our goal in this study is to provide the community a collection of WS descriptions usable as a benchmark, so we decided to focus on the data semantics, too.

There are three ways of performing the annotation process (Benyahia et al., 2009):

- *manual*: the document is analyzed by a specialist in the field
- *automatic*: the task is realized completely by a machine;
- *semi-automatic*: a part is done automatically and the specialist supervises the process.

The WS collection we chose is large enough, so it is critical for us to select a fully automatic tool. The rest of this section is a review and a comparison of the available annotation tools.

Several softwares allow to convert WSDL files to OWL-S files, but without performing any semantic annotation: they only apply a syntactic transformation and present the information contained in the original WSDL file under a form compatible with the OWL-S recommendation. *WSDL2OWLS* is an open source Java application created at the Carnegie Mellon University (Srinivasan, 2004). *OWL-S Editor* is a plug-in for Protégé (itself an ontology development environment) created at SRI (Elenius & Denker, 2006). Another software performing the same task is also called *OWL-S Editor*, but was developed at Malta University (Scicluna et al., 2004).

Radiant is an open source tool created at the Georgia University (Gomadam et al., 2005). It takes the form of an Eclipse plug-in and can output both SAWSDL and WSDL-S files. It provides a GUI which presents the elements constituting the WS description and allows to select the concepts one wants to associate to parameters or operations, by browsing in the selected ontologies. This interface makes the annotation process easier, but the annotation is nevertheless fully manual.

WSMO Studio is another Eclipse plug-in initially designed to edit semantic WS based on the WSMO model. An extension allows annotating WS parameters and operations, and outputting the result under the form of SAWSDL files (Dimitrov et al., 2007). However, the tool does not provide any assistance to the user and the process is fully manual. In fully manual annotation tool category, there is also a Web based application described in (Budinoski et al., 2010).

ASSAM is an open source Java program developed at the University College Dublin (Hess, 2004), able to output OWL-S files. It provides assistance during the annotation process. First, the user starts manually annotating parameters and/or operations using an existing ontology. Meanwhile, ASSAM identifies the most appropriate concepts using machine learning methods. After enough information has been provided, the software is able to propose a few selected and supposedly relevant concepts when the user annotates a new WS.

MWSAF is another open source Java tool created at the Georgia University (Patil et al., 2004). It outputs WSDL-S files, and like ASSAM it has a machine learning capability allowing it to assist the user during the annotation process. It is able to annotate not only parameters and operations, but also non-functional elements.

SAWS (Semantic Annotation of Web Services) (Salomie et al., 2008) relies on a syntactic comparison of parameter and ontological concept names, in order to rank the concepts depending on their relevance. The tool presented in (Bouchiha, 2012) adopts a similar approach, except is uses a semantic distance based on WordNet (Miller et al., 2005). Thanks to this assistance, these tools are qualified of *semi-automatic*. However, the core of the process remains manual, since the user still has to select the concept. Due to our constraint regarding the size of the collection, such an assisted approach does not seem appropriate.

Table 2.3 WS Semantic annotation tools and their properties.

Name	**Output Format**	**Automation**	**Last Update**
Radiant	SAWSDL, WSDL-S	Manual	05/2007
WSMO Studio	SAWSDL	Manual	09/2007
Web-based app.	SAWSDL	Manual	06/2010
ASSAM	OWL-S	Assisted	05/2005
MWSAF	WSDL-S	Assisted	07/2004
SAWS	SAWSDL	Assisted	06/2010
WSDL2OWLS	OWL, RDF	Assisted	06/2005
OWL-S Editor	OWL-S	Assisted	N/A
OWL-S Editor	OWL-S	Assisted	08/2004
Ontology alignment	N/A	Fully automated	08/2011

More recently, fully automated approaches have been developed. The work in (Canturk & Senkul, 2011) is based on ontology alignment. First, WS are thematically grouped together, and associated to a domain ontology, i.e. a predefined ontology describing the main concepts of the identified theme (e.g. heath, education, etc.). Second, for each description file, the service ontology is built by processing the parameter names. Then, concepts are associated to parameters by aligning (i.e. matching) the domain and service ontologies. Although promising, the implementation of this tool is not publicly available, and the authors do not specify which domain ontologies could be used.

From this review, we can conclude the existing annotation tools present various limitations relatively to our goals. First, from a practical perspective, some of these tools are old and not supported anymore, which can cause installation and/or use problems. For instance, Radiant and ASSAM are not compatible anymore with the current versions of some of the Eclipse plug-ins, libraries or API they rely on; meanwhile MWSAF installs and runs fine, but generates files without any of the annotations defined by the user. More importantly, these tools require important human intervention: Radiant and WSMO Studio and the other Web based application are fully manual, whereas ASSAM and MWSAF only assist the user, after a compulsory learning phase. Even if SAWS does not need a learning phase, its capability is not far from ASSAM and MWSAF, in the sense it only assists the user, as WSDL2OWLS, and both OWL-S Editors. The only fully automated tool, which is based on ontology alignment, is not publicly available. Therefore, it is not possible to use it to annotate the syntactic collection we selected. This justifies the development of our own tool, which we present in the next section.

3 PROPOSED METHOD

In previous sections, we showed that there is no appropriate semantic WS collection to test the tools created in the semantic WS area. We explained that it is possible to generate a semantic WS collection from a WS collection by annotating it. As a consequence, we selected the *Full Dataset* WS collection (Hess et al., 2004) as the most representative option. Due to the absence of an appropriate annotation tool, we decided to develop our own software.

Not only we used Full Dataset as raw data to perform the annotation process, but also we benefited from it by identifying patterns consistent with our automatic semantic annotation objective. An iterative process consisting in introducing/removing various processing steps in/from our tool allowed us to determine the best workflow. In this section, we describe our tool in details. We do not give the various modifications tested during its development, but rather focus only on the final version instead. We first introduce its general structure, then its components, detailing their design and functioning.

3.1 General Architecture

MATAWS takes a collection of WSDL files as its input, and outputs the corresponding annotated descriptions under the form of OWL-S files. It was developed in Java, and uses various open source libraries, which are mentioned later. As shown in Figure 3.1, it is made up of three main parts. The first and last parts, *Input Component* and *Output Component,* are dedicated to the processing of the original WSDL files and the generation of the OWL-S files, respectively. The middle part is constituted of several *Core Components* (detailed in Figure 3.2) which implement the semantic annotation itself.

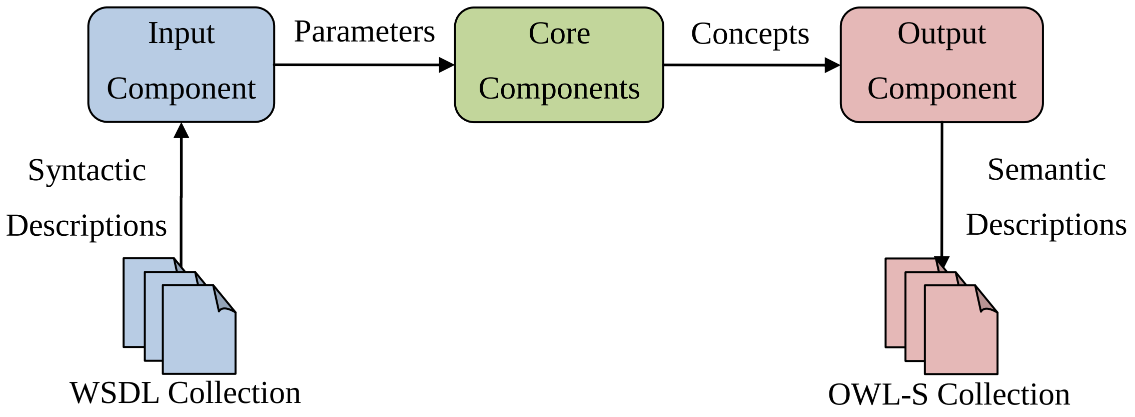


Figure 3.1 General Architecture of MATAWS. The Core Components are detailed in Figure 3.2.

The *Input Component* is responsible for extracting the relevant information from the considered collection of WSDL files. Our annotation process focuses on parameters, which is why we need to retrieve their names, for all the operations defined in the collection. Moreover, in certain cases, we also need to know the name of their data types, and possibly their structure if the type has a complex content. This work is delegated to a parser previously developed by our research team (Cherifi et al., 2011), which produces a set of Java objects representing the extracted information. These objects are fetched to the Core Components, which will associate concepts to the parameters, therefore completing the objects. The *Output Component* is in charge for taking advantage of these completed objects in order to generate a collection of OWL-S files. It relies on an external library called Java OWL-S API (Sirin & Parsia, 2004), which provides a programmatic read/write access to OWL-S files. Our tool can easily be extended to output other semantic WS description formats such as SAWSDL (Farrell & Lausen, 2007), WSDL-S (Akkiraju et al., 2005), provided comparable APIs exist for them.

The organization of the Core Components is represented in Figure 3.2. There are four of them, each one in charge of a specific part of the annotation process. Each parameter is processed separately, through a recursive process. First, the *Preprocessor* receives the Java object representing the parameter, from the Input Component. Its role is to split some name into several words, which are then normalized and filtered. What we call a *name* here is a text string whose meaning is not always directly accessible, e.g. a concatenation of altered words. The Preprocessor is initially applied to the name of the received parameter. It produces a list of words, which is fetched to the next component: the *Selector*. This component is designed to output a single word able to represent the whole list. We call the resulting word the *representative word* for the considered name. Note it can be one of the words from the list sent by the Preprocessor, but it can also be a different, supposedly more meaningful one.

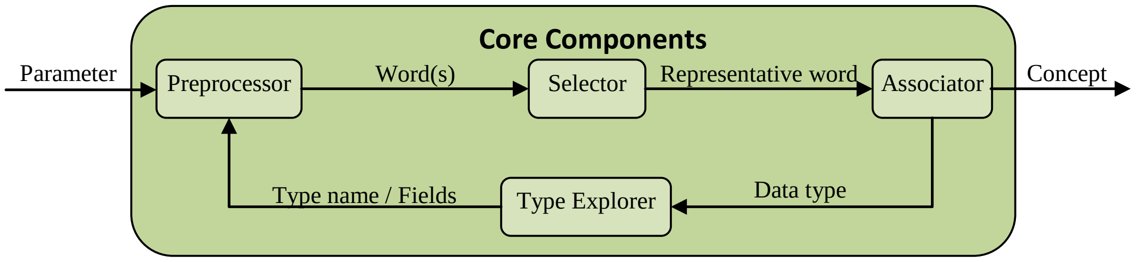


Figure 3.2 Core Components, corresponding to the central part of Figure 3.1.

The next step, performed by the *Associator*, consists in selecting an *ontological concept* able to formally represent the meaning of the representative word. If the Associator is successful and returns a concept for the representative word, this concept is linked to the considered parameter in the corresponding Java object. It can then be used later by the Output Component when generating the OWL-S file.

But, for various reasons explained later, it is also possible that the Associator cannot return a concept. In this case, the *Type Explorer* steps in and retrieves some properties related to the parameter data type. First, the analysis conducted on the parameter name can also be applied to the type name. Indeed, in WS it is common to define custom types, and their name can be sufficiently informative. Second, if the type name is not sufficient to select a concept, then one can consider its structure. For a complex-content type containing fields (i.e. the equivalent of a `struct` type in C programming), the same process can be applied again to each field. Those are sent to the Preprocessor, which can handle them like parameters since they have a name and a type. This results in a recursive process, whose end depends on the selection of a concept by the Associator (success) or the exhaustion of the data type (failure).

The Input and Output Components are not concerned with the business logic of our tool, so they do not require any further description. As a consequence, the following subsections are dedicated to the Core Components only.

3.2 Preprocessor

The Selector and Associator can only process well-formed, clean words. However, the names defined in WSDL files do not usually comply with this requirement, for three main reasons. Firstly, it is difficult to describe the meaning of a parameter, operation or type using a single word. As a result, names are most of the time made up of several concatenated words, separated by alternating lower and upper cases, or by using special characters such as underscores, hyphens, etc. Secondly, sometimes the resulting composite names are too long to be practical, so WS designers use abbreviations in order to reduce their length. Finally, various forms of noise are also present, taking the form of meaningless digits, special characters and spelling errors.

Due to the number of possibilities, it is not possible to determine all forms of names one can find in a WS description. Instead of such an exhaustive approach, we adopted a data-driven method. We manually analyzed the collection of WSDL files selected in section 2.3.2, in order to identify properties and patterns regarding the parameter, type and field names. It became clear most of these names are built using classic programming conventions, such as those applied for the C or Java language. It is therefore possible to define and apply a limited appropriate set of transformations to extract usable words from a name. Our name preprocessing consists of three steps: division splits the name in relevant substrings, normalization turns them into clean words, and filtering remove some irrelevant words. As an example, Figure 3.3 shows the preprocessing of the real-world parameter `PlayersInfoAsString3`, which will be commented in the rest of this section.

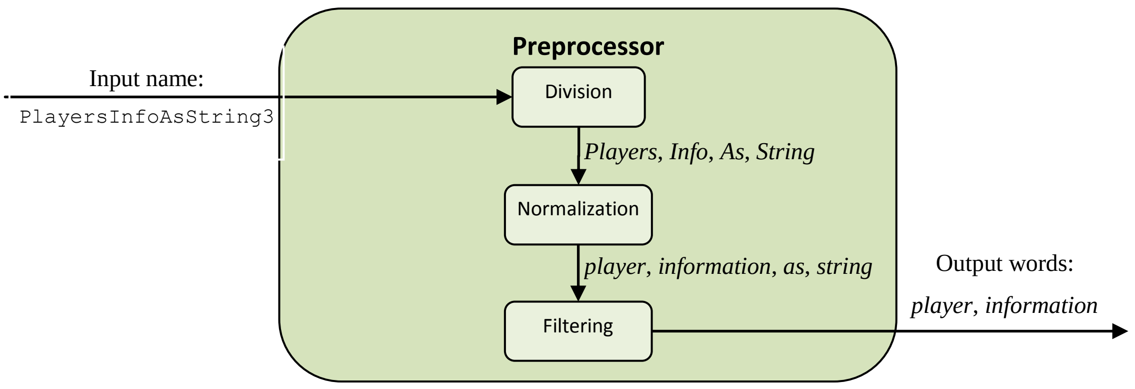


Figure 3.3 Algorithm of the Preprocessor, and application to an example: the parameter name `PlayersInfoAsString3`. The final result is a list of two words: player and information.

The *division* step is mainly responsible for breaking a name into several parts. For this matter, we used several templates presented in Table 3.1. Those were defined after the manual analysis mentioned earlier. They rely mainly on the presence of specific characters or typographic features in the considered names. Additionally, some cleaning is also performed during this step by removing non-letter characters and diacritical marks (accents, cedilla, umlaut, etc.). In the example from Figure 3.3, it takes advantage of the alternation of upper and lower cases to split the name `PlayersInfoAsString3` in 4 parts: `Players`, `Info`, `As` and `String`. The numeric end of the name is removed. However, MATAWS is allowing to split words even in the absence of any typographical information. For instance, we can now break the name `username` down to the two words *user* and *name*. This feature relies on a Java library called *jWordSplitter* (Naber, 2007), which takes advantage of a dictionary to identify words in strings.

Table 3.1 Split templates and examples.

Template	**Name**	**Result**
Upper and lower case alternation	`PlayersInfo`	*Players, Info*
Case alternation with two first case upper	`AParameter`	*A, Parameter*
Underscore separation	`article_id`	*article, id*
Hyphen separation	`date-format`	*date, format*

The *normalization* step corresponds to three tasks, which are all required so that the Associator can efficiently work. The first consists in setting all letters to lowercase. As an example, consider Figure 3.3: all strings lose their initial capital during the normalization. The second task is to replace abbreviations, such as `usr`, with their full length equivalents, such as *user*. Here, a difficulty arises, because some strings can correspond to both a word and an abbreviation at the same time, and/or abbreviate different words. For example, `no` might be used to indicate the negation of the word following it in a concatenated name, such as in `no_limit` (i.e. absence of any limit). But it might also be the abbreviation of the word *number* as in `OrderNo` (i.e. number of the order). The choice of the appropriate meaning is extremely dependent on the considered WS, and could require a human intervention. For this reason, we give the user the possibility to define his own list of abbreviations and corresponding words, as an external text file. We provide a general list of common abbreviations, which can be adapted to various domains of interest. In Figure 3.3, the abbreviation *info* is replaced by the full word *information*. The third and last task consists in replacing a word by its canonical form (i.e. lemma), an operation called stemming. For this matter, we take advantage of the *JAWS* library (Java API for WordNet Searching) (Spell, 2008), which gives a programmatic access to the well-known *WordNet* lexical database for the English language (Miller et al., 2005). WordNet stores the inflectional forms of every word, so JAWS makes it possible to retrieve the canonical form associated to a given inflection. For instance, plural nouns or conjugated verbs are replaced by their singular and infinitive versions, respectively. In Figure 3.3, the string *players* (plural) is replaced by the word *player* (singular).

The role of the *filtering* step is to eliminate stop-words, i.e. words whose meaning is not relevant to our context. For instance, the word *parameter* is largely found in parameter names, but it does not bring any significant information, because we already know if the considered name refers to a parameter. For this reason, it can be regarded as noise and ignored. In Figure 3.3, the words *as* and *string* are considered as stop-words. The result of the preprocessing of parameter `PlayersInfoAsString3` is therefore a list of two words: *player* and *information*. Like for abbreviations, stop-words are extremely context-dependent. For this reason, in MATAWS their specification has been externalized, too. It takes the form of an external text file, initially containing generic stop-words common to most WS. Of course, the user can modify it, in order to adapt it to his own situation.

3.3 Selector

As we explained in the previous section, it is possible to have several words for a parameter name when we realize preprocessing on it. This means that there are several candidate words to annotate for the parameter. However, only one concept can be associated to a parameter in a description file. In order to tackle this problem, we decided to make an inference at this stage, by considering directly the words coming from the Preprocessor. The Selector is in charge of this task, which results in the identification of a single word, called *representative word*. The Associator is then applied to retrieve the corresponding concept, which will be used when generating the semantic description file.

Like for the Preprocessor, we conducted a manual analysis on Full Dataset, the collection of real-world WS descriptions selected in section 2.3.2. This allowed us to identify various cases, which we believe are general enough to be relevant for most WS descriptions. In the rest of this subsection, we first present our analysis and its conclusions, and then describe the algorithm we derived from them.

3.3.1 Data-Driven Analysis

The first goal of our manual analysis was to identify semantic relationships between the various words obtained for a parameter. We preprocessed Full Dataset and found occurrences of synonymy, hypernymy/hyponymy and holonymy/meronymy. We did not look at all for other relationships such as antonymy, because in our opinion, they would not be useful in our context. Two words are *synonyms* if they share a common meaning; this relationship is the only symmetric amongst those mentioned. One word is a *hypernym* of another if its meaning is more general. *Hyponymy* is the inverse relationship: one word is a hyponym of another one if its meaning is more specific. Hyper and hyponymy rely on a general-to-specific definition of inclusion, whereas holonymy and meronymy are whole-to-part based. One word is a *holonym* of another word if the thing it represents contains the thing represented by the second word. The inverse relationship is *meronymy*: one word is a meronym for another one if the thing it represents is contained in the thing represented by the second word. Table 3.2 shows some general examples of such relationships for the word *bus*.

We noticed only a very few cases of synonymial relationships when analyzing Full Dataset, and those appeared to be accidental. Indeed, it seems very unlikely to find several times the same meaning in the description of a single parameter, because it would reflect a redundancy either in the parameter name or data structure. On the contrary, it is possible for the considered words to have several meaning, among which two happen to be similar, leading to a false positive when looking for synonyms. For instance, consider a name such as `coachBus` (meaning: the bus of the -sport- coach). An automatic search would find *coach* and *bus* to be synonyms, whereas in this situation they are not. In order to avoid incorrectly handling this case, which appears to be the most likely in this context, we decided the Selector would not look for synonymial relationships.

Table 3.2 Examples of semantic relationships for the word bus.

Relationship	**Meaning**	**Examples**
Synonymy	Both words have the same meaning	*autobus*, *coach*, *omnibus*
Hypernymy	First word is more general than the second one	*vehicle*, *transport*
Hyponymy	First word is more specific than the second one	*minibus*, *schoolbus*
Holonymy	First thing contains the second one	*fleet*
Meronymy	First thing contained in the second one	*roof*, *window*, *wheel*

The hyper/hyponymy relationship seems much more relevant. When such a direct relationship is detected between two words, we decided to retain the most specific one, because it has a more relevant meaning regarding the context. For example, the name `InterestPercentage` contains *interest* and *percentage*. As shown in Figure 3.4 the former can be considered as the hyponym of the latter, so we can keep it as it is more specialized.

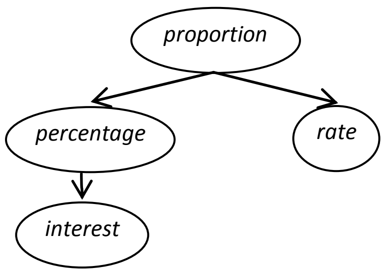


Figure 3.4 Example of semantic relationships between words. The source of an edge is a hypernym, the target is the hyponym.

However, the relationship is not necessarily direct. Let us consider the parameter name `InterestRate`, for instance. As shown in Figure 3.4, on the one hand *interest* is a hyponym of *percentage*, which in turn is a hyponym of *proportion*. On the other hand, *proportion* is also a hypernym of *rate*. Even if *interest* and *rate* are not directly linked by one of the relationships from Table 3.2, they nevertheless share some sense since they have a common hypernym. In this case, one has the choice between three different words: the two original ones, and the common hypernym. It seems more appropriate to retain the latter, since his meaning is supposed to summarize both original words. It is important noticing there is a loss of meaning due to generalization, when replacing a word by a hypernym. In order to limit it, we decided after some experimentation to limit our search for a common hypernym to a distance of only two levels of the semantic tree.

Although looking for common hyponyms is also possible, this approach does not seem to be relevant. First, it is probable that, if two words have one common hyponym, then there are some others, which does not help us since we want to reduce the number of words associated to the parameter. Second, when two words have a common hyponym, it is likely there also exists a direct relationship between them, a case we already tested for. For instance, two synonyms obviously have common hyponyms.

We found two kinds of holo/meronymial relationships in Full Dataset: the direct ones concern lists of words derived from the same name, whereas indirect ones come from several field names, as found in complex-content types. In the first case, let us consider for example a name such as `CarWheel`: *car* is a direct holonym for *wheel*. Most of the time, we found other similarly formed parameter names in the same operation, such as `CarSeat` and `CarEngine`. We chose to keep the meronym, which is more specialized and therefore conveys the most precise meaning; moreover it is the most discriminant part of the name.

In the second case, we have a list of names coming from various fields related to the same parameter. This situation is explained in details in section 3.5, but to summarize: when no concept can be associated to a parameter name or type name, and if the type has a complex content, then its fields are taken into account. Each one is considered as a subparameter and is likely to represent an aspect of the main parameter. Provided a representative word can be selected for each field, they are often indirectly linked via a common holonym. Indeed, this kind of type is relevant to represent the various parts of some object of interest. For example, suppose the considered type contains some fields whose representative words are *handlebar, mudguard* and *pedal*. One of these words holonyms is *bicycle,* which is likely to be the representative word of the parameter.

But for most of the names, there is no obvious semantic relationship between the words extracted during the preprocessing. It is therefore necessary to propose another way of identifying a representative word. In the simplest case, the combination of the extracted words actually forms a compound word, or a well-known expression, for which we can find a single word synonym. For example, consider the parameter name `LastName`, which leads to the two words *last* and *name*: then *surname* constitutes a one word synonym for this compound. The expression *postalcode* (two words *postal* and *code*) can be summarized by the single word *postcode*.

In the case where the extracted words cannot be summarized, our analysis showed one of the extracted words is generally a noun whose meaning is central, and which is complemented by the other extracted words. It seems difficult to automatically retrieve the overall meaning of the expression. But it is worth noticing the central noun alone conveys the most important part of this meaning. Therefore, it constitutes a good representative word, even if part of the meaning is lost in the operation. In the English language, this central word is generally located at the end of the expression, which makes it easy to identify automatically. For instance, in the parameter names `BillingCountry`, `AuthorizationNumber`, `ApplicationName`, `AdvertTypeID` and `AdminEmail`, the central (and therefore representative) words are *country*, *number*, *name*, *id* and *email*, respectively. Note this case is very similar to the direct holo/meronymial relationships described earlier (cf. example `CarWheel`), except here we do not need to identify a direct semantic relationship. If there is no noun at all amongst the extracted words, then there is usually at least one verb. We consider it as the central word, since the rest are adjectives or adverbs, by definition. We consequently select it as the representative word. If several verbs are present, it is difficult to identify the most important, relatively to the considered context. The most objective approach is then to the keep the verb the most frequently used in the English language.

3.3.2 Procedure Design

Based on our previous observations, we derived an algorithm aiming at obtaining the representative word for some list of words extracted during the preprocessing. It takes the form of a series of independent steps, represented in Figure 3.5. For each one of them, the goal is to reduce the number of words in the list, while minimizing the overall information loss. This can be realized either by suppressing some words considered irrelevant or neglectable, or by replacing several words by a new one, which supposedly summarizes them. In the latter case, it is necessary to start the whole process all over again, since the new word might trigger different conditions. This explains why the diagram presented in Figure 3.5 contains loops. As soon as only one word remains in the list, it is considered as the representative word and the process is over.

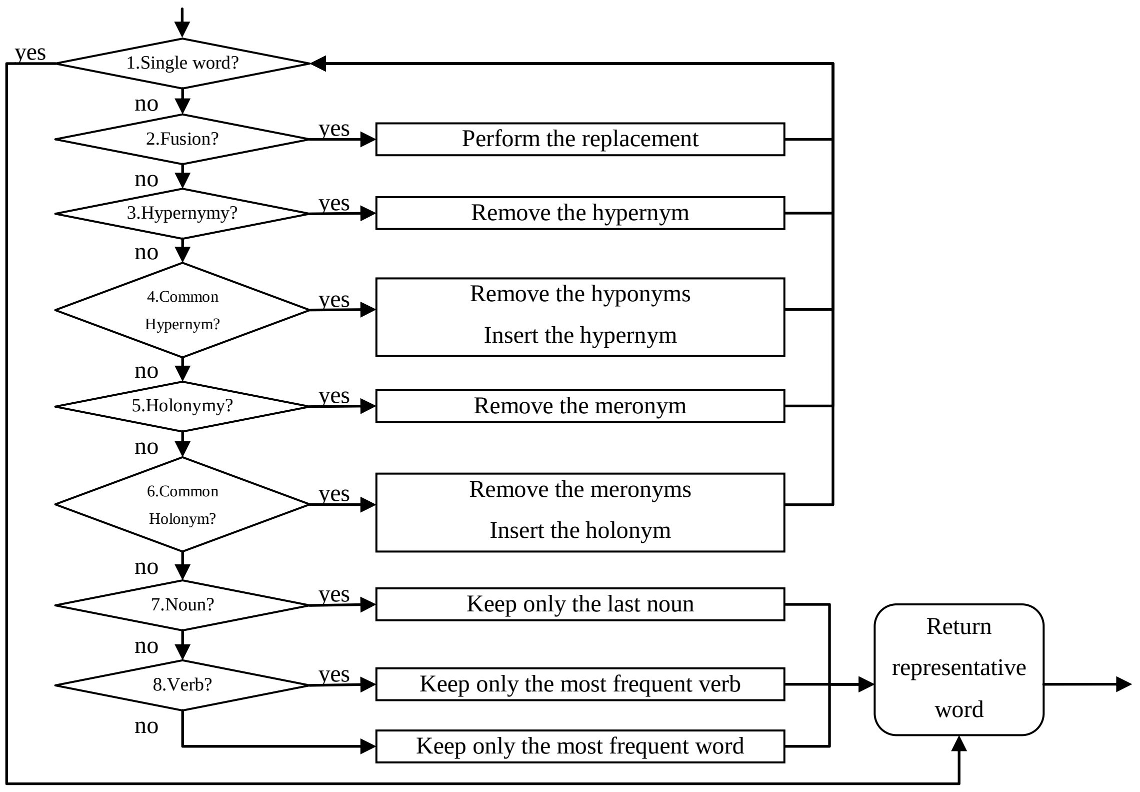


Figure 3.5 Algorithm of the Selector, with examples.

Our process relies largely on WordNet (Miller et al., 2005), accessed through the JAWS API (Spell, 2008), already mentioned for the stemming operation in the preprocessor. First, WordNet contains the necessary semantic relationships described in Table 3.2. But we also use it to determine the grammatical nature of a word (verb, noun, etc.). Indeed, WordNet is able to provide all the grammatical roles associated to the various meanings of a given word. For instance, the word *clean* has 1 meaning for which it is considered as a *noun*, 10 meanings for which it is a *verb*, 18 for which it as an *adjective*, and 2 for which it is an *adverb*. For our purpose, we use an approximation by retaining the most frequent role of the word, based on the assumption it is the most likely to occur in our situation, too. Finally, the last advantage of WordNet is its compatibility with Sigma (Pease, 2008), the tool at the core of the Associator (described in section 3.4). As mentioned before, Sigma maps WordNet to SUMO (Niles & Pease, 2001), which means all the words obtained through JAWS can be associated to an ontological concept.

The different steps of the algorithm are ordered depending on how much of the original information they allow retaining: we check first the most favorable situations, and finish with the cases corresponding to the most approximate operations. Suppose we receive the following list of words as an input of the Selector: *customer*, *mailing*, *address*, *kitchen*, *lavatory*, *washroom*, *postal*, *code* (this example is purely artificial, it was defined for illustration purposes only). The 1st step is the stop condition, met when there is less than one word in the list. In the 2nd step, we use WordNet to check if certain words can be identified as a single expression. If it is the case, the concerned words are replaced by the expression in the list, and the process starts again. In our example, *postal* and *code* can be replaced by *postcode*, so the updated list is *customer*, *mailing*, *address*, *kitchen*, *lavatory*, *washroom*, *postcode*.

The 3rd step is concerned with direct hyper/hyponymial relationships. In our example, *mailing* and *address* correspond to this kind of relationship because *mailing* is a hypernym of *address*. As explained before, only the most specific word (the hyponym) is kept, so here we remove *mailing* and get the list *customer*, *address*, *kitchen*, *lavatory*, *washroom*, *postcode*. In the 4th step, we look for indirect hypernymial relationships.

The word *bathroom* is a common hypernym for *lavatory* and *washroom*, so they are both replaced by *bathroom*, which summarizes them. The list thus becomes: *customer*, *address*, *kitchen*, *bathroom*, *postcode*.

The 5th step is dedicated to the detection of direct holo/meronymial relationships. In our list, *address* and *postcode* have this relationship, since *postcode* is a part of *address*, i.e. *postcode* is a meronym of *address*. In this case, we keep the meronym, i.e. *postcode*. The remaining words are therefore *customer*, *kitchen*, *bathroom*, *postcode*. The 6th step corresponds to the processing of words coming from different fields, i.e. we look for common holonyms. The words *kitchen* and *bathroom* correspond to parts of *home*, which is therefore a common holonym. In the list, we replace both words by *home*, obtaining: *customer*, *home*, *postcode*.

If the remaining words are not connected by any direct or indirect semantic relationships, we reach the last steps, which rely on grammatical information. The 7th one tries to detect nouns. In our case, we have three of them: *customer*, *home* and *postcode*: as explained earlier, we make the assumption the last ones are complements of the one placed just before, as in “the postcode of the customer’s home”. We therefore keep *postcode*. We only have a single word remaining in the list, so the process is complete and *postcode* is our representative word. We therefore do not reach the 8thstep, which focuses on verbs.

If after all the previous steps, there are still several words in the list, then we choose the most frequent one (in the English language) amongst them.

3.4 Associator

Thanks to the process performed by the Selector, the Associator receives only a single word for each parameter. Its role is then to identify the most appropriate ontological concept to represent in a formal way the meaning of this word. For this purpose, we employ Sigma, which is a Java API implementing various ways of creating, testing, modifying and inferring on ontologies (Pease, 2008). It is bundled with SUMO

(Suggested Upper Merged Ontology) (Niles & Pease, 2001), the largest formal public ontology available up to now. It was first built using the SUO-KIF language (IEEE, 2003), but it is now also available in OWL (McGuinness & Harmelen, 2004), a language compatible with OWL-S (Martin et al., 2004), the current output format of MATAWS.

Table 3.3 Examples of concept associations.

Word	**SUMO Concept associated by Sigma**
buffalo	`HoofedMammal`
school	`EducationalProcess`
talk	`Communication`

Sigma is mainly designed for working on ontologies but it also implements a mapping between WordNet and SUMO, which is particularly interesting for us. Indeed, the words coming from the Selector have been outputted by JAWS, and are therefore contained in WordNet: this means Sigma should be able to find an ontological concept for each the word the Associator receives. Table 3.3 gives a few examples of the concept associations returned by Sigma. It is important to note the mapping does not necessarily associate a concept of the same semantic level: it is often more general than the original word, like for example *buffalo* and `HoofedMammal` in Table 3.3. Compared to an automatic approach such as ontology alignment (Canturk & Senkul, 2011) (cf. section 2.3.3), such a mapping has the advantage of having been defined and verified manually. It encodes the knowledge of experts, and is therefore more reliable than an algorithmic approach when it comes to identifying the ontological concept corresponding to a word.

Various methods give a programmatic access to the mapping, taking English words as input and returning SUMO concepts as output. In particular, for a given word, Sigma is able to return a list of concepts whose meaning can be expressed through the considered word. The reason of this multiplicity comes from the fact a word can have several distinct meanings. However, the way this list is ordered is arbitrary: it depends on the indexation order of the concepts in Sigma's database, and not on any relevance criterion. In order to tackle with this point, we take advantage of the word usage frequency information available in WordNet. It allows us to identify the most frequent meaning of the word, and then select the concept corresponding to this specific meaning, amongst those returned by Sigma. Therefore the concept returned by the Associator is more likely to correspond to the actual meaning of the word.

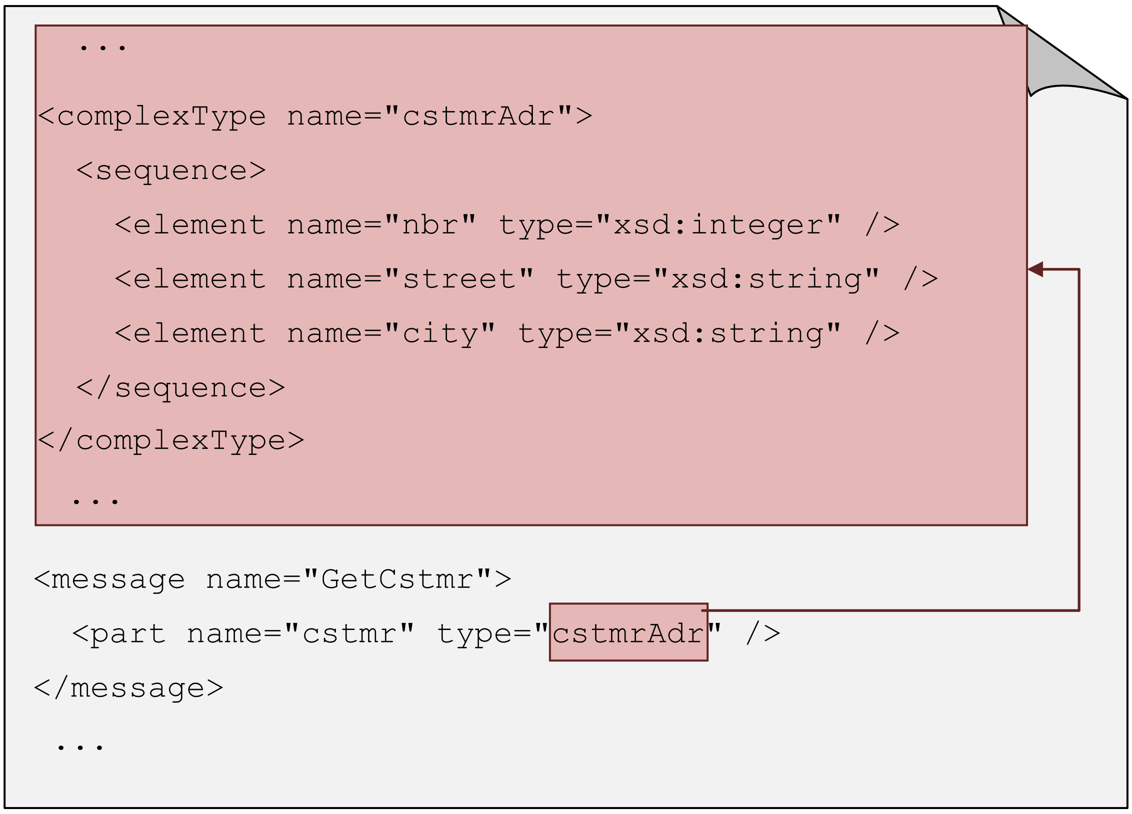
3.5 Type Explorer

Most of the time, the process implemented by the components described in the previous sections is sufficient to associate a concept to a parameter. However, there are some specific situations for which they do not succeed. First the Preprocessor might fail to break the name down to relevant words. This can be due, for instance, to an atypical way of forming the name, or even to errors in their definition, e.g. `mypaRametr`. This is likely to prevent the Associator from identifying a concept. Second, if the parameter name is a stop word, or contains only noise and stop words, e.g. `AParameter_11`, then the Preprocessor will completely filter it. The Selector will therefore receive an empty list, and will not be able to identify a representative word to fetch the Associator. Third, for some reason which was not identified yet, the Associator very rarely fails to return a concept for certain words, even if they are correct English words.

Under one or several of these circumstances, the Associator is not able to return a concept for the considered parameter. Yet, the identification of such a concept is of course necessary to annotate the WS description. In order to overcome this problem, we take advantage of the latent semantics possibly conveyed by the data type of the parameter. It takes two different forms: first the name of the type, and second its complex content. This means our method cannot be applied to primitive types (`integer`, `string`, etc.). In this case, MATAWS is definitely unable to identify a relevant concept for the considered parameter. This is signaled by associating a special symbol `NoMatch` representing the absence of concept in the generated OWL-S file. If the type is custom, it is likely to bear an explicit name, i.e related to the information contained represented by the parameter. This name can be treated using the exact same process already applied to the parameter name.

But since the process is the same, it can fail for the same reasons described earlier. In this case, the content of the type can be analyzed, provided the type has a `sequence` complex content. As mentioned in section 2.1.3.2, such a type allows one element to contain a series of other elements. The parameter type is therefore very much like a structured type in C-like programming languages, i.e. it is made up of several fields. Note that for now we focus on `sequence` complex content because it is the most widespread, however the same approach can be extended to the other kinds of XSD complex content as well.

The fields have themselves a name and a data type, so our process can be applied on each of them separately. In the best case, if the field names are sufficient, this results in a list of representative words: one for each field. Those can then be combined by the Selector to get a single representative word for the whole parameter. If one (or more) name turns out to be insufficient, its type name can be used like we did before for the parameter. And if the type name is still not enough, then its (possibly) complex content can be ultimately exploited. This result in a recursive process: as long as the element data type has a complex content, the process can be applied one step deeper, until the considered type has a simple content.



```
  ...
<complexType name="cstmrAdr">
  <sequence>
    <element name="nbr" type="xsd:integer" />
    <element name="street" type="xsd:string" />
    <element name="city" type="xsd:string" />
  </sequence>
</complexType>
  ...

<message name="GetCstmr">
  <part name="cstmr" type="cstmrAdr" />
</message>
 ...
```

Figure 3.6 Excerpt from a real-world WSDL file, representing a parameter with complex-content XSD type.

Figure 3.6 displays a part of a real-world WSDL file, in order to illustrate the notion of complex-content type. A parameter named `cstmr` has a type called `cstmrAdr`. This type has a complex content consisting of a sequence of two strings: `name` and `address`. The intended parameter and type names are `customer` and `customerAddress`, respectively. However, their meaning cannot be directly retrieved automatically, because the actual names are not explicit enough. Since a first pass is not enough to determine the associated concept, Mataws will consider the type content. The names of the elements in the sequence would lead to the words *number*, *street*, and *city*, which the Selector would have to combine, in order to obtain a single representative word for the parameter, e.g. *address*.

4 RESULTS

We tested MATAWS by applying it to the collection of real-world WS descriptions selected in section 2.3.2. But this collection was not used only for evaluation, also for driving the development of MATAWS. In order to show whether MATAWS is collection dependent or not, we additionally decided to apply it to another collection. So we prepared a new WS collection, we called *Control collection*, by collecting real-world WS descriptions publicly and individually available on the WWW. Then we applied MATAWS to the control collection, too.

The result of the automatic semantic annotation of WS descriptions can be evaluated in two different ways. We can obviously adopt a *quantitative* perspective, and consider the proportion of parameters for which MATAWS was able to provide an ontological concept. But this type of evaluation is relatively limited, because even if one concept was associated to a parameter, this concept can be completely irrelevant. In the case of a real-world application, suppose the human user performs a second pass after MATAWS, in order to manually annotate the parameters for which MATAWS failed to identify a concept. Then, providing an incorrect concept is worse than providing none at all, because the user will not detect and correct these errors. It is therefore necessary to complete the quantitative evaluation with a *qualitative* one. This operation can be performed only manually, since it consists in trustworthily annotating a consequent number of parameters and comparing the resulting concepts with those automatically selected by MATAWS.

In this section, we first present the two WS collections used during evaluation of MATAWS by giving their main characteristic features. Then we pass to the evaluation phase; first we will show our quantitative results of the first and second collections separately, then we proceed similarly with the qualitative results, before finally discussing them.

4.1 Data

As we mentioned in the previous section, we have two collections to evaluate MATAWS. The first collection, *Full Dataset* from the ASSAM project (Hess et al., 2004), was used during the development of MATAWS, and we previously mentioned it in our review of WS descriptions collections (section 2.3.2). It contains 7877 operations distributed over 816 real-world WS descriptions. Moreover, it has 9869 parameter instances in total, corresponding to 2465 unique parameters. We consider two instances correspond to the same parameter if they both have the same name and data type (so, independently from the operation they belong to). The ways how we benefited from this collection during the development of MATAWS are detailed in sections 3.2 and 3.3. We apply the final version of MATAWS to this collection in order to see whether the tool we developed is efficient.

Before looking at the result evaluation of the Full Dataset, we know that it may not be sufficient to indicate MATAWS as successful even if it shows a good performance on this collection. Because we developed it under the light of this collection and there may be a collection dependence. For this reason, we prepared a control collection, consisting of real-world WS descriptions found on the Internet. We gathered to them using the Google search engine. This collection has 100 services and totally 1473 operations are found on them. In addition, the control collection contains 5476 parameter instances in total, of which 1695 are unique parameters. We apply MATAWS also to this collection in order to show that MATAWS is general enough and can work for every collections.

In Table 4.1, we summarize the features of these two collections. We pass to the evaluation of our results in the following sections.

Table 4.1 Comparison of the features of evaluation collections

Collection Name	Number of Service	Number of Operations	Number of Parameters	
			Total	Unique
Full Dataset	816	7877	9869	2465
Control collection	100	1473	5476	1695

4.2 Quantitative Results

Since MATAWS outputs either a single concept or no concept at all for each parameter, we consider a parameter as *annotated* if MATAWS is able to return a concept.

To perform the evaluation, we considered the rates of annotated parameters relatively to both views (instances and unique parameters). The unique parameters rate represents how well the software performs when considering the heterogeneity of parameters. The parameter instances rate is more descriptive of the performance regardless how parameters are distributed in the collection.

Table 4.2 displays both annotation rates for the Full Dataset collection. The difference of 19.85 points observed between parameter instances and unique parameters rates is due to the fact the parameters the tool could not annotate are amongst the most frequent in the collection.

Table 4.2 Proportions of annotated parameters on Full Dataset

Data	Total	Annotated	Annotation rate
All Parameter Instances	9869	7542	76.42%
Unique Parameters Only	2465	2373	96.27%

During the development process of MATAWS, it evolved to handle all possible kinds of parameter names using the Preprocessor. It was enriched with new features in every versions such that in the final version, apart from single names or names that have a separator between their words, MATAWS can handle parameter names without separator such as `emailaddress` or `filedata`, or names taking a non-canonical form, such as `allocated` (conjugated verb) or `weeks` (plural noun) as we mentioned in section 3.2. However, a non-neglectable proportion of parameters still remains non-annotated. These correspond mainly to names containing only stop-words, such as `Body` and `Return`, and associated with simple-content data types. Put differently, the parameters whose names and types do not convey sufficient information remain impossible to annotate. We call them *meaningless parameters*. With that in mind, we consider the performance of MATAWS on Full Dataset to be satisfying enough.

Table 4.3 shows the annotation rates for the Control collection, in a way similar to Table 4.2 for Full Dataset. On unique parameters, MATAWS annotated almost the same proportion (-1.52 points) of the collection, whereas on parameter instances it is even higher ($+13.39$ points). Therefore, MATAWS displays good generalizing features, at least on these results, and seems fairly collection-independent. The difference of 4.94 points observed between parameter instances and unique parameters rates remembers us of the same behavior observed for Full Dataset. Again, the parameters the tool could not annotate are amongst the most frequent in this collection. However, we can say that this kind of parameters are less frequent in the Control collection, so the effect on the performances is much lower.

Table 4.3 Proportions of annotated parameters on the Control collection

Data	**Total**	**Annotated**	**Annotation rate**
All Parameter Instances	5476	4918	89.81%
Unique Parameters Only	1695	1606	94.75%

Studying the amount of annotated parameters allows us to evaluate the final output of MATAWS. But it is also possible to consider the representative words outputted by the Selector, which can be considered as partial results produced during the annotation process. This gives us an insight of the internal behavior of our tool. Since only one representative word is outputted for each parameter, the numbers of word instances in Table 4.4 and Table 4.5 are the same than the numbers of parameter instances in Table 4.2 and Table 4.3.

Table 4.4 Proportions of annotated words on Full Dataset.

Selected	Total	Annotated	
All Word Instances	9869	7542	76.42%
Unique Words Only	623	613	98.39%

Table 4.4 displays the proportions of annotated words from the Full Dataset collection, including the case where no word at all could be fetched to the Associator. The tool behaves like for the parameters, since its rate undergoes an increase of the same order than before (+21.97). Further analysis reveals 2307 out of these 9869 cases are completely filtered by the Preprocessor and Selector (i.e. the Associator receives an empty string). They correspond to meaningless parameters, such as `return`, `parameter` or `x`. Without them, the proportion of annotated word instances would be 99.73% (instead of 76.42%). Getting rid of such parameters is actually a good thing, because even if the Associator is generally able to identify a concept for them, it is most of the time completely irrelevant: this results in a lowered annotation quality, as discussed in the next subsection. For the unique words, we obtain a high score of 98.39%, which means almost all the words reaching the Associator are annotable.

Table 4.5 Proportions of annotated words on the Control collection

Selected	Total	Annotated	
All Word Instances	5476	4918	89.81%
Unique Words Only	478	460	96.23%

As seen on the Table 4.5, the results obtained for the Control collection present the same behavior than for the parameters. There is an increase in the unique words rate ($+6.42$), like on the Full Dataset. While the annotation rates are almost the same (-2.16) in unique words for both collections, more word instances ($+13.39$ points) are annotated in the Control collection. The difference is similar to what was observed with the parameter instances results.

4.3 Qualitative Results

The qualitative evaluation basically consists in grading the quality of the concepts outputted by MATAWS, in terms of relevance with the original parameters. This operation must be conducted manually, since it is the only means we have to assess relevance.

The first method that comes to mind is to directly compare the final output of MATAWS (i.e. a concept) with the input parameter. However, considering the representative words, stays appealing because they allows a separated evaluation of the two semantic-related components of MATAWS: the Selector and the Associator. This is doubly interesting, because the former was developed totally by us and reflects more our efforts, whereas the latter is based upon an external tool.

In consequence, we finally applied the following two-stepped procedure. For each parameter, we first considered the representative word outputted by the Selector, and manually assigned a score expressing its relevance relatively to the parameter. We then similarly assigned a score to the concept outputted by the Associator, expressing its relevance to this representative word. For this matter, we took advantage of all available data: not only the parameter and data type names, but also contextual information such as the WS textual description or operation name. The first score allows us to evaluate how good the Preprocessor and Selector perform, whereas the second one stands for the performance of the Associator.

In order to consider a parameter has been successfully processed, we consequently need both scores to be high. Indeed, if the first is low, it means the representative word is not relevant, and thus even if the final concept is a good match for this word, it will fail to appropriately represent the original parameter. If the first score is high, but the second is low, then it means the tool failed to associate a relevant concept to an appropriate representative word, leading again to a incorrect result. The whole tool can consequently be evaluated by considering the minimum of both scores.

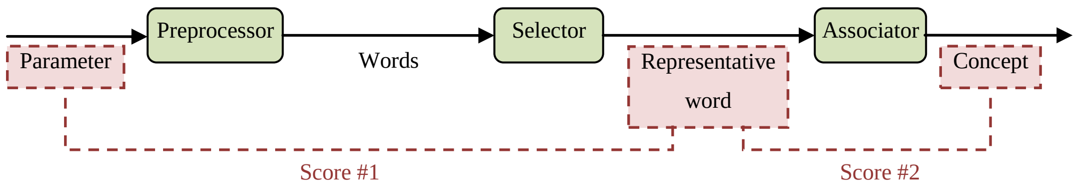


Figure 4.1 Representation of evaluation method used for MATAWS. The Type Explorer is not represented for readability matters. It is possible to get two scores: the first one is obtained by comparing the initial parameter with its representative word, and the second one by comparing this word with the associated concept.

To be more precise and account for ambiguous cases, we chose to adopt a multivalued score ranging from 0 (not relevant at all) to 5 (completely relevant). The score measures two different aspects of the annotation: correctness and precision. With the former, we aim at quantifying how much the associated meaning (representative word or concept) is semantically related to the considered object (parameter or representative word). The latter accounts for the fact the associated meaning sometimes subsumes the actual one. This is often the case for the concepts outputted by the Associator, due to the nature of mapping implemented by Sigma (see section 3.4). Figure 4.1 summarizes the evaluation procedures of MATAWS in a comprehensive way.

Let us first consider the performance at the level of the parameters for the Full Dataset collection, displayed in Table 4.6. As mentioned before, there are three different scores for the second version of MATAWS: relevance of the representative word relatively to the original parameter (noted *P vs. W*), relevance of the ontological concept relatively to the representative word (*W vs. C*) and relevance of the ontological concept relatively to original parameter (*P vs. C*). The latter is the minimum of the two former, and corresponds to the overall performance. All values displayed in Table 4.6 are average scores. Like before, we processed separately the results for parameter instances and for unique parameters. Unlike with the quantitative evaluation, and for all scores, the results are better for parameter instances than for unique parameters: $+0.34$, $+0.16$ and $+0.36$, respectively. This indicates the tool is good on the most frequent annotated parameters (in terms of the quality of annotation, this time).

Table 4.6 Average parameter annotation scores on Full Dataset.

Data	**Annotated**	**P vs. W**	**W vs. C**	**P vs. C**
Annotated Parameter Instances	7542	4.18	3.45	3.04
Annotated Unique Parameters Only	2373	3.84	3.29	2.68

The following remarks hold for both parameter instances and unique parameters. The best score is obtained for *P vs. W*, which means the representative words outputted by the Selector are semantically very close to the meaning of the corresponding parameters. The score decreases when considering *W vs. C*, which reflects the fact the Associator is not able to find a relevant concept for a part of the representative words. The overall score *P vs. C* is lower, meaning some of the representative words successfully annotated by the Associator were actually not relevant to the original parameter. The standard deviation for this last score is close to 2, so the somewhat neutral (in the sense it is close to $2.5/5$) average score hides the fact parameters are associated to either very relevant or very irrelevant concepts.

Table 4.7 Average parameter annotation scores on control collection.

Data	**Annotated**	**P vs. W**	**W vs. C**	**P vs. C**
Annotated Parameter Instances	4918	4.08	3.28	2.86
Annotated Unique Parameters Only	1606	3.81	3.02	2.55

Let us now consider the results obtained on the Control collection. As seen on Table 4.7, our tool presents a similar behavior: for all scores, the results are better for parameter instances than for unique parameters: $+0.27$, $+0.26$ and $+0.31$, respectively. Moreover, these differences have the same order of magnitude than what was observed for Full Dataset. Compared to the scores obtained on the Full Dataset collection, we get also very similar scores: the differences between the two collections are -0.1, -0.17, -0.18 for parameter instances and -0.03, -0.27, -0.13 for unique parameters. This confirms the observation we made in the quantitative evaluation section, regarding the fact MATAWS does not seem to be collection-dependent.

Table 4.8 Average word annotation scores on Full Dataset.

Data	**Annotated**	**W vs. C**
Annotated Word Instances	7542	3.45
Unique Annotated Words Only	613	2.74

We now focus on the assessment of representative words for the Full Dataset collection, shown in Table 4.8. Like for the parameters, the average score increases when comparing words instances to unique words ($+0.71$). Thus, similarly to what was observed for parameters, the tool is better with frequently annotated words.

Table 4.9 Average word annotation scores on control collection.

Data	**Annotated**	**W vs. C**
Annotated Word Instances	4918	3.28
Unique Annotated Words Only	460	2.81

For the Control collection, and like on the parameters, our tool gives results which are relatively similar to those obtained on the Full Dataset (see Table 4.9). The score of word instances is better than the score of unique words (+0.47). Therefore, the same remark than before holds: the tool performs a better annotation of the frequent words.

Table 4.10 Qualitative results for the top 15 most frequent parameters on Full Dataset: frequency (number of occurrences of the parameter), corresponding representative word (the word itself and its associated score), ontological concept (concept and score) and overall score.

Parameter	**Freq.**	**Representative Word**		**Ontological Concept**		**P vs. C**
		Word	**P vs. W**	**Name**	**W vs. C**	
`ApplicID`	228	*identity*	5	`TraitAttribute`	3	3
`Password`	223	*password*	5	`LinguisticExpression`	4	4
`UserID`	148	*identity*	5	`TraitAttribute`	3	3
`password`	114	*password*	5	`LinguisticExpression`	4	4
`username`	85	*user*	3	`experiencer`	4	3
`AdminUserID`	75	*identity*	5	`TraitAttribute`	3	3
`Result`	68	*integer*	4	`Integer`	5	4
`LicenseKey`	58	*key*	5	`Key`	5	5
`strGuidNotification`	58	*notification*	5	`RegulatoryProcess`	3	3
`UserName`	58	*user*	3	`experiencer`	4	3
`IsReleased`	52	*release*	3	`Demonstrating`	0	0
`accession`	46	*accession*	5	`Increasing`	0	0
`EmailAddress`	44	*address*	3	`uniqueIdentifier`	4	3
`MaxRecords`	40	*record*	3	`Text`	1	1
`date`	39	*date*	5	`Day`	4	4

In order to give a more meaningful example of the results outputted by MATAWS, Table 4.10 presents the obtained for the 15 most frequent parameters in Full Dataset. The results obtained for the parameters ending in `-ID` (`ApplicId`, `UserID`, `AdminUserID`) demonstrate the interest of the Selector, leading to the representative word *identity*. The parameter `Result` illustrates the notion of meaningless parameter introduced in section 4.2. It is associated to the word *integer* and to the concept `Integer`. Its name is considered as a stop word in our context, and it has a simple data type. As explained before, the best MATAWS can do in this situation is to take advantage of the type name.

The differences observed between the representative word and concept scores justify the necessity to distinguish the Selector and the Associator in terms of performance. The former extract a relevant representative word for most of the listed parameters: *identity* for `ApplicID`, *notification* for `strGuidNotification`, *key* for `LicenceKey`, etc. It is not as clear for the latter: `TraitAttribut` for *identity*, `experiencer` for *user* and `Key` for *key* seem fairly relevant, but `LinguisticExpression` and `RegulatoryProcess` for *password* and *notification*, respectively, seem too general to be close enough to the actual meanings. This is due to the fact the mapping implemented by Sigma sometimes associates an ontological concept which is not directly equivalent to the considered word, semantically, but rather subsumes it, as explained in section 3.4. Some concepts are completely irrelevant, e.g. `Increasing` for *accession* and `Demonstrating` for *release*. These problems could be solved by replacing Sigma by a similar but more precise tool. However, such software, which would implement an equivalent mapping from the English language to an ontology, does not exist to the best of our knowledge. It seems more realistic to keep Sigma as the base of our Associator, and refine its results through some additional processing based on complementary NLP tools.

Table 4.11 Qualitative results for the top 15 most frequent parameters on Control collection: frequency (number of occurrences of the parameter), corresponding representative word (the word itself and its associated score), ontological concept (concept and score) and overall score.

Parameter	Freq.	Representative Word		Ontological Concept		P vs. C
		Word	P vs. W	Name	W vs. C	
password	516	password	5	LinguisticExpression	4	4
username	464	user	3	experiencer	4	3
id	217	identity	5	TraitAttribute	4	4
description	65	description	5	Stating	1	1
in	62	in	5	Inch	0	0
out	62	out	5	ContestAttribute	0	0
name	61	name	5	ContentBearingObject	3	3
plan	56	plan	5	Plan	5	5
session	35	session	5	FormalMeeting	4	4
type	34	type	5	subclass	5	5
snumber	28	s	0	SecondDuration	3	0
login	26	gin	0	DistilledAlcoholicBeverage	5	0
application_key	26	key	5	Key	5	5
Result	25	response	3	causes	0	0
Status	25	status	5	SubjectiveAssessmentAttribute	3	3

We present the 15 most frequent parameters of the Control collection in Table 4.11, in order to compare the outputs of both collections from this point of view. We observe that the Preprocessor steps in twice with its split feature for the case where no separator are found between words of the parameter name (for parameters `snumber` and `login`). Splitting `snumber` into `s` and `number` is appropriate whereas dividing `login` into `lo` and `gin` indicates a problem. This problem happens rarely when the parameter name has more than one option to split. That is why we kept this feature of the Preprocessor. Apart from them, the representative word extraction results are very good and similar to what was obtained on Full Dataset. Like in the previous table, the results for word-to-concept association are not stable: `Plan` for *plan* and `subclass` for *type* show good examples of concept association, whereas `ContestAttribute` for *out* and `causes` for *response* refer to more general meanings regarding the (supposed) intended meanings of the words. This reminds a previous remark again: evaluating the Selector and the

Associator should be done separately. As a consequence, the results of concept association to parameters are very similar for both collections. Therefore, the most frequent parameters aspect seems to indicate, again, that our tool performance is independant from the data.

5 CONCLUSION

In this thesis, we presented our tool MATAWS, which can semantically annotate syntactic WS descriptions in a fully automatic manner. The process is based on the mapping of the WordNet (Miller et al., 2005) lexicon to the SUMO ontology (Niles & Pease, 2001) implemented in Sigma (Pease, 2008), on the Selector, whose role is to identify the most relevant word, relatively to a given parameter, amongst the list outputted by the Preprocessor, and on a multimodal approach consisting in taking advantage not only of the parameter names, but also of their data type names and structures.

We applied MATAWS to ASSAM Full Dataset (Hess et al., 2004), the largest available collection of WSDL files, and evaluated the resulting collection of OWL-S files. The proportion of annotated parameters exceeds 75% of the collection. It does not cover all parameters, however further analysis of the results shows the non-annotated parameters would be hard to process even to humans, due to the lack of context (e.g. parameters simply called `parameter`). Thus, we can say that MATAWS is successful from the quantitative perspective. When considering the quality of the annotation, i.e. the relevance of the concepts associated to the parameters, our tool has an average grade exceeding 3/5. Comparing to the Associator, the Selector component has more contribution on this average score: the analysis of partial results extracted during the process shows the semantic process it implements has a direct effect on the relevance of the outputted concepts. Therefore, we can accept that MATAWS is successful also from a qualitative point of view.

Since MATAWS was developed under the light of Full Dataset, we prepared a new control collection from real-world WS descriptions found on the Web, and we applied MATAWS to this collection too, in order to prove that its results are not collection-dependent. The results obtained show an obvious parallelism with those from Full Dataset. This seems to indicate the performances of MATAWS are relatively stable, independently from the collection.

The first contribution of our work is the MATAWS tool. This was the object of an article accepted and presented at the Networked Digital Technologies (NDT 2011) conference (Aksoy et al., 2011b). The second contribution is the result of the annotation process: a collection of semantic WS descriptions, which can be used as a benchmark to test some of the many methods developed specifically to take advantage of this semantic aspect. Both the tool and the collection were the object of an article submitted to Elsevier's Journal of Web Semantics, which is currently being reviewed (Aksoy & Labatut, 2013). Moreover, note the tool is open source, relies on freely available libraries, and is freely available itself[5]. The files in the produced semantic collection follows the OWL-S format (Martin et al., 2004), a W3C-supported technology, and the collection is itself publicly available. It is bundled with a table containing the detail of the manual annotation work we performed when evaluating our tool.

As we mentioned earlier, it seems difficult to increase the performance of MATAWS relatively to the proportion of annotated parameters. However, there is room for improvement regarding the quality of the annotation. Our experimental evaluation showed the main problem comes from the Associator, which does not always pick the most relevant concept when a word can be associated to several ones. We see two possible, non-mutually exclusive solutions to this problem: first, take advantage of some additional information to improve the concept selection; and second, use a different method to retrieve the concept. MATAWS does not use certain parts of the WSDL files, such as the names of messages or operations, and the optional natural language descriptions (see section 2.1.3.2). Those could be exploited directly, like we did for parameter names and types. Another possibility is to adopt the approach seen in (Canturk & Senkul, 2011), and use them to identify the domain ontology of the WS. This would allow the Associator to work on a subset of concepts, and therefore decrease

[5] http://code.google.com/p/mataws/

the risk of selecting an irrelevant one. Regarding the second solution, potential alternatives to Sigma exist, although their use is generally less direct, such as DBPedia[6] or Wikipedia-based dictionaries like the one described in (Spitkovsky & Chang, 2012). Comparing and combining the results outputted by several such tools constitutes a promising perspective.

[6] http://wiki.dbpedia.org

REFERENCES

Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M. T., Sheth, A., Verma, K. (2005). Web Service Semantics - WSDL-S. URL: http://www.w3.org/Submission/WSDL-S/. [accessed December, 2012].

Aksoy, C., Mancuhan, K. (2010). Annotation Automatique de Descriptions de Services Web. BSc, Galatasaray University, Istanbul, TR.

Aksoy, C., Labatut, V., Cherifi, C., Santucci, J.-F. (2011a). MATAWS: A Multimodal Approach for Automatic WS Semantic Annotation. Communications in Computer and Information Science, 136 (6), p.319-333.

Aksoy, C., Labatut, V., Cherifi, C., Santucci, J.-F. (2011b). MATAWS: A Multimodal Approach for Automatic WS Semantic Annotation. In: 3rd International Conference on Networked Digital Technologies, Jul 11-13, Macau.

Aksoy, C., Alparslan, E., Bozdağ, S., Çulhacı, İ. (2011c). OSDBQ: Ontology Supported RDBMS Querying. In: Metadata and Semantic Research (MTSR 2011), October 12-14, Izmir.

Aksoy, C., Labatut, V. (2013). A Fully Automatic Approach to the Semantic Annotation of Web Service Descriptions. Journal of Web Semantics, Submitted.

Benyahia, K., Lehireche, A., Latreche, A. (2009). Annotation Sémantique de Pages Web. In: CIIA, May 3-4, Saida.

Berners-Lee, T. (1998). Weaving the Web. San Francisco: Harper Eds.

Berners-Lee, T., Hendler, J., Lassila, O. (2001). The Semantic Web. Scientific American, p.34-43.

Biron, P. V., Malhotra, A. (2004). XML Schema Part 2: Datatypes Second Edition. URL: http://www.w3.org/TR/xmlschema-2/. [accessed November, 2012].

Bouchiha, D. (2012). Semantic Annotation of Web Services. In: 4th International Conference on Web and Information Technologies, April 18-21, Porto.

Brickley, D., Guha, R. V. (2004). Official site of W3C for RDF-S. URL: http://www.w3.org/TR/rdf-schema/. [accessed December, 2012].

Budinoski, K., Jovanovik, M., Stojanov, R., Trajanov, D. (2010). An Application For Semantic Annotation Of Web Services. In: 7th International Conference for Informatics and Information Technology, February 3-6, Skopje.

Cabral, L., Domingue, J., Motta, E., Payne, T., Hakimpour, F. (2004). Approaches to Semantic Web Services: An Overview and Comparisons. LNCS, 3053, p.225-239.

Canturk, D., Senkul, P. (2011). Semantic Annotation of Web Services with Lexicon-Based Alignment. In: IEEE World Congress on Services, Jul 4-9, Washington.

Chauvet, J.-M. (2002). Services Web avec SOAP, WSDL, UDDI, ebXML. Paris: Eyrolles.

Cherifi, C., Rivierre, Y., Santucci, J.-F. (2011). WS-NEXT, A Web Services Network Extractor Toolkit. In: 5th International Conference on Information Technology, May 11-13, Amman.

Christensen, E., Curbera, F., Meredith, G., Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. URL: http://www.w3.org/TR/wsdl. [accessed December, 2012].

Clement, L., Hately, A., Riegen, C., Rogers, T. (2005). Official site of OASIS for UDDI. URL: http://www.oasis-open.org/specs/index.php#uddi. [accessed December, 2012].

De Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Keller, U., Stollberg, M. (2005). Web Service Modeling Ontology (WSMO). URL: http://www.w3.org/Submission/WSMO/. [accessed December, 2011].

Dimitrov, M., Simov, A., Momtchev, V., Konstantinov, M. (2007). WSMO Studio - A Semantic Web Services Modelling Environment for WSMO (System Description). LNCS, 4519, p.749-758.

Elenius, D., Denker, G. (2006). OWL-S Editor. URL: http://owlseditor.semwebcentral.org/index.shtml. [accessed December, 2011].

Fallside, D. C., Walmsley, P. (2004). XML Schema Part 0: Primer Second Edition. URL: from http://www.w3.org/TR/xmlschema-0/. [accessed November, 2012].

Farrell, J., Lausen, H. (2007). Semantic Annotations for WSDL and XML Schema. URL: http://www.w3.org/TR/sawsdl/. [accessed December, 2012].

Gomadam, K., Verma, K., Brewer, D., Sheth, A., Miller, J. (2005). Radiant: A Tool for Semantic Annotation of Web Services. In: International Semantic Web Conference, November 6-10, Galway.

Gruber, T. (1993). A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2), p.199-220.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., Lafon, Y. (2007). Official site of W3C for SOAP. URL: http://www.w3.org/TR/soap. [accessed December, 2012].

Hess, A. (2004). ASSAM (Automated Semantic Service Annotation with Machine Learning) WSDL Annotator. URL: http://www.andreas-hess.info/projects/annotator/index.html. [accessed December, 2011].

Hess, A., Johnston, E., Kushmerick, N. (2004). ASSAM: A Tool for Semi-Automatically Annotating Semantic Web Services. In: International Semantic Web Conference, November 7-11, Hiroshima.

Hyvönen, E. (2002). Vision, Technologies, Research and Applications. In: Semantic Web Kick-Off in Finland, November 2, Helsinki.

IEEE. (2003). SUO-KIF (Standard Upper Ontology Knowledge Interchange Format). URL: http://suo.ieee.org/SUO/KIF/suo-kif.html. [accessed October, 2012].

Kellert, P., Toumani, F. (2004). Les Web Services Sémantiques. Interaction Intelligence Information, 3 (1), p.77-102.

Klyne, G., Carroll, J. J. (2004). Official site of W3C for RDF. URL: http://www.w3.org/TR/rdf-concepts/. [accessed December, 2012].

Kökciyan, N. (2009). Classification de Services Web Par des Méthodes de Clustering. BSc, Galatasaray University, Istanbul, TR.

Kreger, H. (2001). Web Services Conceptual Architecture. URL: http://www.cs.uoi.gr/~zarras/mdw-ws/WebServicesConceptualArchitectu2.pdf. [accessed November, 2012].

Küster, U., König-Ries, B., Krug, A. (2008). OPOSSum - An Online Portal to Collect and Share SWS Descriptions. In: 2th IEEE International Conference on Semantic Computing, August 4-7, California.

Ma, J., Zhang, Y., He, J. (2008). Web Services Discovery Based on Latent Semantic Approach. In: International Conference on Web Services, September 23-26, Beijing.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Sycara, K. (2004). OWL-S: Semantic Markup for Web Services. URL: http://www.w3.org/Submission/OWL-S/. [accessed December, 2010].

McGuinness, D. L., Harmelen, F. V. (2004). Official site of W3C for OWL. URL: http://www.w3.org/TR/owl-features/. [accessed December, 2012].

Miller, A. G., Fellbaum, C., Tengi, R., Langone, H., Ernst, A., Jose, L. (2005). Wordnet. URL: http://wordnet.princeton.edu/. [accessed December, 2011].

Naber, D. (2007). jWordSplitter. URL: http://jwordsplitter.sourceforge.net/. [accessed November, 2012].

Niles, I., Pease, A. (2001). Towards a Standard Upper Ontology. In: International Conference on Formal Ontology in Information Systems, October 17-19, Ogunquit.

OASIS. (2004). Introduction to UDDI: Important Features and Functional Concepts. URL: http://uddi.org/pubs/uddi-tech-wp.pdf. [accessed November, 2012].

Paolucci, M., Sycara, K., Kawamura, T. (2003). Delivering Semantic Web Services. In: Twelves World Wide Web Conference (WWW2003), Budapest, Hungary.

Patil, A., Oundhakar, S., Sheth, A., Verma, K. (2004). METEOR-S Web service Annotation Framework. In: International Conference on the World Wide Web, May 17-22, New York.

Pease, A. (2008). Sigma Knowledge Engineering Environment. URL: http://sigmakee.sourceforge.net/. [accessed September, 2012].

Rivierre, Y. (2010). Discovery and Composition in Web Services Networks. BSc, Université Joseph Fourier, Grenoble, FR

Ryman, A. (2003). Understanding Web Services. URL: http://www.ibm.com/developerworks/websphere/library/techarticles/0307_ryman/ryman.html. [accessed December, 2012].

Salomie, I., Chifu, V. R., Giurgiu, I., Cuibus, M. (2008). SAWS: A Tool for Semantic Annotation of Web Services. In: IEEE International Conference on Automation, Quality and Testing, Robotics, May 22-25, Cluj-Napoca.

Scicluna, J., Abela, C., Montebello, M. (2004). Visual Modelling of OWL-S Services. In: IADIS International Conference WWW/Internet, March 24-26, Madrid.

Sheth, A. P. (2003). Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration. In: Workshop on E-Services and the Semantic Web, May 20, Budapest.

Sirin, E., Parsia, B. (2004). The OWL-S Java API. In: International Semantic Web Conference, November 7-11, Hiroshima.

Skoutas, D., Sacharidis, D., Kantere, V., Sellis, T. K. (2008). Efficient Semantic Web Service Discovery in Centralized and P2P Environments. In: International Semantic Web Conference, September 23-26, Beijing.

Spell, B. (2008). Java API for WordNet Searching. URL: http://lyle.smu.edu/~tspell/jaws/index.html. [accessed October, 2012].

Spitkovsky, V. I., Chang, A. X. (2012). A Cross-Lingual Dictionary for English Wikipedia Concepts. In: 8th International Conference on Language Resources and Evaluation, May 21-27, Istanbul.

Srinivasan, N. (2004). WSDL2OWL-S. URL: http://www.semwebcentral.org/projects/wsdl2owl-s/. [accessed December, 2011].

BIOGRAPHICAL SKETCH

Cihan Aksoy was born in 1987 in İstanbul, Turkey. He finished Vehbi Koç Vakfı High School in 2005 and received his Bachelor's degree of Computer Engineering in 2010 from the Galatasaray University in Istanbul, Turkey. He is currently pursuing a Master's degree in Computer Engineering at the same university, while working at TUBİTAK SGE as a researcher.

His first publication is a conference paper written under the supervision of the Asst. Prof. Dr. Vincent Labatut. Its title is "MATAWS: A Multimodal Approach for Automatic WS Semantic Annotation" (Aksoy et al., 2011b), and it was presented at the Networked Digital Technologies (NDT 2011) international conference. He wrote another publication related with "Ontology Supported Audit" project realized for the Turkish Court of Accounts at TÜBİTAK SGE. Its title is "OSDBQ: Ontology Supported RDBMS Querying" (Aksoy et al., 2011c) and it was presented at the Metadata and Semantics Research (MTSR 2011) conference. Finally, he has written an extended and improved version of his first paper, entitled "A Fully Automatic Approach to the Semantic Annotation of Web Service Descriptions" (Aksoy & Labatut, 2013), which was submitted to Elsevier's Journal of Web Semantics and is currently under review.