An Approach to Extend WSDL-Based Data Types Specification to Enhance Web Services Understandability

By
Fuad Sameh Ali Alshraideh

Supervisor
Dr. Samer Hanna

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer Science

Deanship of Academic Research and Graduate Studies
Philadelphia University

May 2013

جامعة فيلادلفيا
نموذج تفويض

أنا فؤاد سامح علي الشريدة ، أفوض جامعة فيلادلفيا بتزويد نسخ من رسالتي للمكتبات أو المؤسسات أو الهيئات أو الأشخاص عند طلبها.

التوقيع:
التاريخ:

Philadelphia University
Authorization Form

I am, Fuad Sameh Ali Alshraideh, authorize Philadelphia University to supply copies of my thesis to libraries or establishments or individuals upon request.

Signature:
Date:

An Approach to Extend WSDL-Based Data Types Specification to Enhance Web Services Understandability

By
Fuad Sameh Ali Alshraideh

Supervisor
Dr. Samer Hanna

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer Science

Deanship of Academic Research and Graduate Studies
Philadelphia University

May 2013

Successfully defended and approved on May, 29, 2013

Examination Committee	**Signature**	
Dr. Academic Rank:	, Chairman.	_ _ _ _ _ _ _ _ _ _ _ _
Dr. Academic Rank:	, Member.	_ _ _ _ _ _ _ _ _ _ _ _
Dr. Academic Rank:	, Member.	_ _ _ _ _ _ _ _ _ _ _ _
Dr. Academic Rank: ( )	, External Member.	_ _ _ _ _ _ _ _ _ _ _ _

Dedication

I fully dedicate this thesis:

To my father "Sameh Alshraideh"

And my mother "Khawla Alshraideh"

And my great wife "Wedad"

And my lovely son "Qusai"

For being the most important part of my dream

For the support, courage, and unconditional love.

Fuad Alshraideh

Acknowledgment

Thanks TO ALLAH first, before and after everything, for giving me the knowledge and ability to complete this work in this final form.

I would like to thank our university for offering scientific nutrition necessary to complete our study and our research. I would like to express my sincere thanks to the staff of the college who provide a warm and lively environment to encourage and help graduate students in their graduate study. Especially, Prof. Saeed AL-Goul, Dr. Nameer Al-Emam, Dr. Moayad Al-Athami, and Dr. Wael Hadi for their support of the educational process.

I would like to extend my regards and sincere gratitude to Dr. Samer Hanna, who has guided me through my work from the beginning and supported me and for being a treasure of knowledge and moral.

I would like to thank my brothers (Mohammed, Khaled, Ahmad) and my sisters(Malak, Haifa, Hala) for their unconditional love and support, and never forget to thank my brother's wife Carolyn Miller and her mother Suzan Miller, and Dr. Doniazad Alshraideh for their great support.

Lastly, I would express my great thanks to my best friend Ra'ed Alazaidah and all of my friends at Philadelphia University.

Fuad Alshraideh

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Meaning
CIM	Computation Independent Model
HTTP	Hypertext Transfer Protocol
IDL	Interface Definition Language
MDA	Model Driven Architecture
OWL	Web Ontology Language
PSM	Platform Specific Model
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
WSDL	Web Service Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

Abstract

Web Services are important for integrating distributed heterogeneous applications. One of the problems that facing Web Services is the difficulty for a service provider to represent the datatype of the parameters of the operations provided by a Web service inside Web Service Description Language (WSDL).

This problem will make it difficult for service requester to understand, reverse engineering, and also to decide if Web service is applicable to the required task of their application or not.

This thesis introduces an approach to extend Web service datatypes specifications inside WSDL in order to solve the aforementioned challenges. This approach is based on adding more description to the provided operations parameters datatypes and also simplified the WSDL document in new enrichment XML-Schema.

The main contributions of this thesis are:

1. Comprehensive study of 33 datatypes in C# language, and how they are represented inside WSDL document.
2. Classification of the previous mentioned datatypes into 3 categories: (Clear, Indistinguishable, and Unclear) datatypes.
3. Enhance the representation of 18% of C# datatypes that are not supported by XML by producing a new simple enrichment XML-based schema.
4. Enhance Web Service Understandability by simplifying WSDL document through producing summarized new simple enrichment shcema.

CHAPTER ONE

INTRODUCTION

When talking about the connection of heterogeneous systems through network (Internet or intranet), it must be remembered that these applications, endpoints, or systems are exchanging data via messages (R.Virgilio et al, 2010) to perform various operations over a network. But why do systems exchange data across a network? And what is the nature of these exchanges? Certainly, users of these systems are looking for the implementation of various operations they cannot perform on their systems, so they seek to carry out these tasks or operations on other systems connected on the same network. But how can one identify the systems that have the ability to provide such operation? What are the main roles in these exchanges? Is there any impact for the datatype on the exchanges? All of these questions are related to the concept of Web services, around which this thesis is stationed.

1.1 Research Problem

One of the problems of Web service is the difficulty of a service provider to represent the data types of parameters of the operations provided inside Web Services Description Language (WSDL). This problem makes it difficult for a service requester to understand, reuse, reverse engineer, integrate and compose Web Services, and also to decide if a Web service is applicable to the required task of their application. The major question for this thesis is that : Can we extend the datatypes specifications inside WSDL to reach for a better comprehension of the WSDL documents by the requesters and providers of the Web Services ?

1.2 Why this problem ?

Web Service is a new paradigm for building distributed software applications that have many advantages, such as increasing the interoperability between heterogeneous applications and usability, reusability and also deployability (S. Hanna et al, 2010). In order to achieve this goal, Web Services paradigm depends on many Extensible Markup Language (XML) based standards such as Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), and Universal Description Discovery and Integration (UDDI), which will be illustrated in depth in chapter two. These standards support interaction, description and selection of Web Services. WSDL documents use XML Schema Datatypes (XSDs) to specify the input and output parameters datatypes for operations provided by a Web Service.

WSDL documents contain many challenges and problems such as a complexity because it contains a large amount of information that make it difficult to be read. The problem of interest to this thesis is that the (XSD) system, which is used to specify the allowed datatypes of a certain Web Service, is not expressive enough to represent and specify many of the Web Services operations input and output datatypes (S. Hanna et al, 2011) .

Many commonly used simple datatypes are built into XML Schema, however, some datatypes which are important to the current programming languages are not supported by XSD. Consider the following example that illustrates this problem:

Suppose that Service provider application contains the following operation signature in Figure (1-1).

```
public char charExample (Char charPart)
```

Figure (1-1): A sample signature of an operation.

Service Providers will find it difficult to describe this operation using WSDL because the datatype system used by WSDL, namely XSD, does not include the *char* datatype.

Web Service developers depend, most of the time, on Web Service development tools such as WCF and Axis2 to generate the description of their services represented by WSDL documents. The result of this experiment has shown that for the operation signature in Figure 1.1, the *char* datatype was presented inside WSDL as follows:

1- Using Windows Communication Foundation (WCF) :

```
<xs : Element name=“charExample”>
  <xs :  complexType>
    <xs : sequence>
      <xs : element xmlns:q10 = htt ://schemas .microsoft.com/2003/10/Seria lization/,minOccurs=“0”Name=“charPar” type=“q10:char”/>
    </xs : sequence>
  </xs : complexType>
</xs : element>
```

2- using Java :

```
<xs : complexType name=“charExample”>
  <xs:sequence>
    <xs : element name= “arg0 type=“xs : unsignedShort” minOccurs=“0”/>
  </xs : sequence>
</xs : complexType>
```

3- Using Axis2 :

```
<element name-"charExample">
  <complexType>
    <sequence>
      <element name=:"charPar" Type="xsd : anyType"/>
       </sequence>
    </complexType>
 </element>
```

This example illustrates the inconsistency problem of the Web Service development tools in specifying Web Service operations parameters datatypes. By using WCF tool the *char* datatype is defines inside WSDL as *(<xs:element xmlns:q10 = htt ://schemas .microsoft.com/2003/10/Serialization/,minOccurs="0"Name="charPar"type="**q10:char**"/>)* the parameter *char* is a custom datatype which defined by the tool itself, it has *char* type and it is defined inside WSDL as *q10:char* .By using Java tool the parameter (*arg0*) which has the same datatype *char* defined inside WSDL as *(<xs : element name= "**arg0 type="xs : unsignedShort**" minOccurs="0"/>),* as we noted that it is defined as *unsignedShort*, not as *char* and also by using Axis2 it is defined the parameter (*charPar*) as *(<element name=:"charPar" Type="xsd : anyType"/>)* with *anyType* instead of *char* type. This simple example for primitive datatype *char* gives quick overview of the problem which we will attempt to solve, and in chapter 4 we present a full discussion and more examples about datatypes implementation inside WSDL .

It has also been noticed that after examining many real WSDL documents and other experimental WSDLs, that different tools will provide different representations of the same datatype inside WSDL, and this will decrease the understandability of the Web Service by its requesters and also developers. This problem is the main focus of this thesis, and we will illustrate our proposed approach to deal with this problem in chapter 4.

1.3 Research Contributions

In this thesis, we aim to meet the following objectives:

1. Finding the information that must be added to the WSDL in order to generate a better specification of the Web services input and output messages operations parameters datatypes.

2. Investigating the effect of the Web Service development technique on the input and output messages operations parameters datatypes inside Web Service Definition Language (WSDL) .

3. Building a Computer Aided Software Engineering (CASE) tool that can automatically generate a more understandable specification or representation to the Web service input and output messages operations parameters datatypes, and which will support the development process of the Web Services through the creation of the models at high level of abstraction.

4. The automatic generation of clearer and more understandable datatypes specifications for Web Services.

1.4 General structural Design

The general process for the proposed tool in extending the XML-Schema datatype specification which we will use in this thesis is shown in five stages. Firstly, the tool can extract the WSDL document which attached with the Web Service itself, and then extract the Data type parts on which we want to do our extendsion. By adding annotation and more description and also constraints facets to the Datatypes part the result is a new enrichment WSDL document with more specific Datatypes. Then new UML will be generated depending on the new enrichment WSDL document, which has more detail about the Datatypes used with the Web Service, and also gives the client a clear and simple description for Web Service without any ambiguity.

1.5 Thesis Outline

The rest of this thesis consists of 5 chapters, organized as follows: Chapter two discusses the main components of Web Service and explains them in detail (such as XML-Schema ,WSDL ,UDDI ,SOAP), Chapter 3 reviews general approaches used to implement and to describe the functionality of Web Service which the client uses to decide if the Web Service is applicable for his needs. Chapter 4 and Chapter 5 presents proposed method and how to implement our tool on any Web Service. Chapter 6, summarizes the main achievements of this thesis, presents the general conclusions and suggests further research directions.

1.6 Summary

In this chapter, we have introduced a brief introduction on Web Service. We also discussed the importance of the Web Service for integrating distributed heterogeneous applications. One of the problems that is facing Web Services is the difficulty for a service provider to represent the parameter datatypes of the operations provided by a Web Service inside the WSDL. We proposed an approach based on adding more description to the provided operations parameters datatypes and also enabling the service provider to produce a document that describes the datatypes in an understandable form.

CHAPTER TWO

BACKGROUND

Chapter one provided a brief, general introduction about Web Services. As previously shown, we are interested in enhancing the understandability of the datatype specification inside WSDL and also will give a full and accurate explanation for all Web Service parts to further clarify the main idea of this thesis. In this chapter we will give a full explanation of all web service components and also full explanation for its main roles.

2.1 What are Web Services ?

A review of the various studies showed that a large number of definitions for Web Service have been proposed. For example (H. EL Bouhissi et al, 2009) defined the Web Service as software components that allow access to functionality via a Web interface network. Additionally, (H. Wang et al, 2004) and (J. Rao et al, 2004) defined it as a software system designed to support machine-to-machine interaction over a network. These brief definitions detail a new breed of Web applications with self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. According to this thesis, Web Services are defined as a collection of applications (interface application) or a collection of systems (endpoints) interacting with each other by exchanging data and information over networks. Each service has its self-located, self-describing and also self-operational properties.

If one of these endpoints is to provide service over network (Internet or intranet), then the provider must publish a full and detailed explanation for this service. This detailed explanation is called Web Service Description Language (WSDL) (H. EL Bouhissi et al, 2009) and (R. Virgilio et al, 2010). WSDL makes it easier for other endpoints which share the same network to know more about the provided service, and then to decide if this service is applicable for their needs or not.

There are three roles that can be defined within an interaction with Web Service, as follows:

1. **Service Provider**: The application or the system which introduces the service (J. Jiang et al, 2005). This part plays the major role in a Web Service. When anyone wants to publish a programmed service over network, he must initially program this service using one of the programming languages such as Java, C# or other programming languages that allow the creation of such software (J. Fu

et al, 2011). The aim of such software services is to facilitate interaction between clients and Web Services by making them independent from the technologies with which they have been implemented (W. Sun et al**,** 2009), and also to transfer interaction over network as application-to-application. In this area it should also be said that these services providers cannot publish their services over the network unless they publish a full description called Web Service Description Language (noted earlier), which is automatically generated through the programming language tools such as Apache Axis or .NET (R. Grønmo et al, 2004), used to generate these Web Services itself.

2. **Service Requester**: This is the second most important part of the Web Service, defined as client or user looking for an application or any type of operation that he cannot get or apply through his system, so he seeks it via network by using one of the web browsers to find the suitable published service which implements all of his requirements. (W. Sun et al**,** 2009) and (J. Jiang et al, 2005). In this way, the user can decide whether the service is suitable for his requirements.

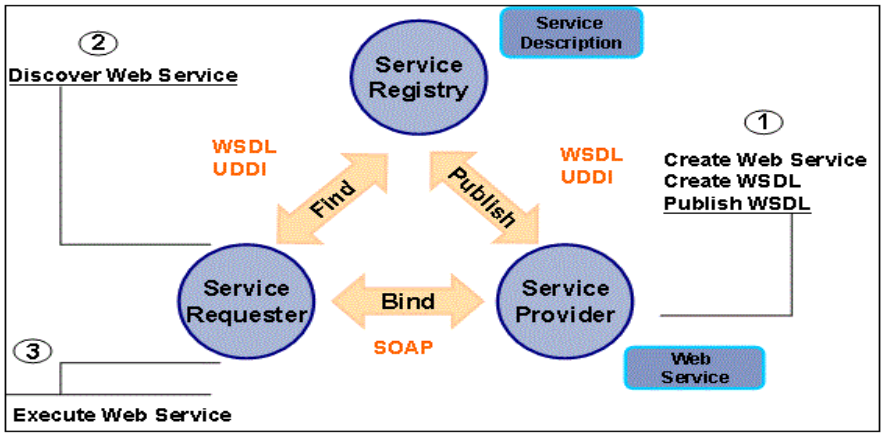
   The initial and basic step requires the requester to look for the WSDL document that is always attached with the published service. This document has all the information necessary to help the requester decide about the applicability of the Web Service for his needs (A. Bellini et al, 2010).

3. **Service Registry**: Web Services are web components that are only accessible via interfaces published in standard Web services-specific interface definition languages and accessible via standard network protocols (J. Zhang et al, 2005). The third role of the Web Service is the registry application, which is an interface created by the service provider itself (J. Fu et al, 2011). It is generated in order to give the user a way to bind or interact with service, to benefit from the services provided. A high level of facilities and techniques are presented to the users or clients to enable integration of the Web Services (J. Hendrik et al, 2005).

   Ultimately, the user can decide if the service is appropriate for his requests. If it is appropriate then he fills the Application with all necessary data, and then the service becomes available for him and can apply whatever he wants remotely, by

sending his request message (XML-Based message) and then receiving the response message from the Web Service itself through the network (E. Domínguez et al, 2007) and (J. Hendrik et al, 2005) .

As stated in the previous paragraphs, the three roles can be defined within an interaction with web services shown in Figure (2-1).



http://www.jot.fm/issues/issue_2005_09/column2/

Figure (2-1): Web Service roles

2.2 Web Service Components

Web Services are functional components available over the internet. These components are primary technologies that have emerged as worldwide standards that make up the core of today's Web Services technology. All of the Web Service components are XML-Based and we will explain all of these components next. Figure (2-2) shows Web Service components.

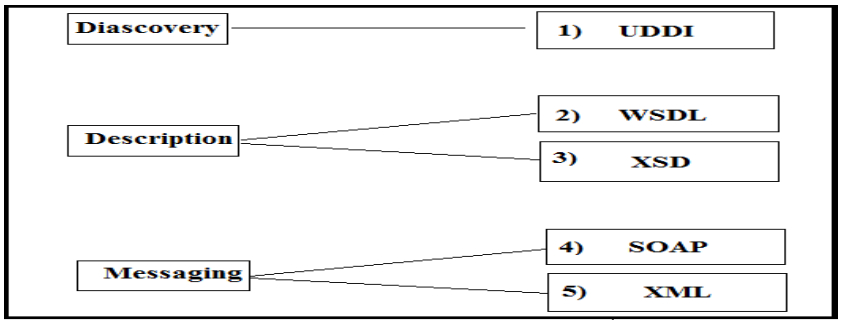


Figure (2-2): Web Service Components

2.3 Web Service Advantages and Challenges

Web Services are becoming increasingly important for integrating service-oriented applications (W. Sun et al**,** 2009) because of the provided increased interaction between applications and systems, and also heterogeneous systems over network. Thus, we cannot ignore the role Web Service plays within the concept of distributed systems, so we should mention some of the most prominent features (S. Hanna et al, 2010):

1. Increasing the reusability of the Web Services and accordingly reducing the time and cost required to build a Web-based distributed application.
2. Facilitate and streamline communication and interactions between systems and heterogeneous applications over network.
3. Providing the ability for interoperability between various software applications running on irregular platforms/operating systems.
4. Based on open standards and protocols.

However, Web Service faces numerous challenges and problems, including, but not limited to the following (S. Hanna, 2008) :

1. **The trustworthiness problem**: The Service Requester can only see the contract (WSDL) of a Web Service but not the source code. This fact has caused Service Requesters to question the trustworthiness of Web Service because Service Requesters do not trust Web Services that were implemented by others without seeing the source code. (W. Tsai et al, 2005) mentioned that this problem is limiting the growth of Web Service applications and that these applications will not grow unless researchers meet this trustworthiness challenge. (J. Zhang, 2005) stated that the current methods and technologies cannot ensure Web Service trustworthiness and that for Web Services to grow, researchers must address this challenge.

2. **The selection problem:** Service Requesters have no criteria to choose between Web Services that accomplish the same task. (J. Zhang, 2005) stated that it is a big challenge to choose the most appropriate Web Service from a "sea of unpredictable Web Services". The reason for these problems and challenges is that the WSDL contract of a Web Service describes the operation or the

function that a Web Service provides and how to bind to this Service. However, it does not describe the non-functional quality attributes such as robustness, reliability or performance.

3. **Vulnerability to invalid inputs by malicious Service Requesters**: Since Web Services are advertised in the Internet, any Service Requester can access this Web Service and some of these might be malicious Requesters that aim to do harm. The Web Input manipulation vulnerability is 59.16% of the overall Web Services vulnerabilities (W. YU et al, 2006) and that is why Web Services should be tested against this kind of fault to assess if a Web service is vulnerable to input manipulation attacks in order to increase Web service trustworthiness. (G. Myers, 1979) mentioned that testing that a program does what it is supposed to do is only half the battle, the other half is to test whether the program does what is not supposed to do. In other words, to check if a program is vulnerable to invalid input.

2.4 What does XML mean?

The main goal of distributed systems is to make the exchangeability of data documents over networks between different applications easy and accessible to all. But what would happen if each application used binary form (machine-readable) encoding for its own data document? The exact details of encoding were often a proprietary standard and unbounded, so that it is difficult for other applications to read or process this document (C. Lee et al, 2009). As a result it is not easy to exchange data files.

Computer scientists have worried for decades about the lack of compatibility and interchangeability of data documents. There has been ongoing movement toward developing a common data document format to become standard practice. This provides conformity to ensure interoperability between applications (C. Lee et al, 2009). This new standard-Based language is called SGML (Standard Generalization Markup Language). Later, W3C developed a subset of it to get new language which is XML (Extensible Markup Language) (A. Bunduchi et al, 2004.) .

Web Services implement Service Oriented Architecture (SOA) based on Extensible Markup Language (XML-Based) open standards such as Simple Object Access Protocol (SOAP), WSDL, and also UDDI. This means Web Services structure is divided into three fields: communication protocol, service discovery and also service description (H. Wang et al, 2004) . These fields will be explained in depth in next sections.

2.5 Web Service Description Language (WSDL)

WSDL documents are used to describe Web Service specifications such as location, functionality, datatypes of the input and output parameter of the operations which this Web Service provides (S. Hanna et al, 2010). To describe the datatypes, WSDL documents depend on the XML-Schema datatype (XSD) system provided by W3C and it will be discussed later.

Web Services Description Language (WSDL) is inherently complex and difficult to understand, even for developers. This difficulty and complexity in understanding WSDL is greatly interesting to researchers (W. Sun et al**,** 2009). As stated previously, WSDL defines Web Services as a network of ports that exchange messages between each other as request and response (Requester, Provider) to get port types which are groups of operations. The data format specification and concrete protocol for these ports must be subjected to binding reuse.

Both WSDL and also XML-Based description allow a structured way to standardize the description of Web Service, and it is also the major part of our thesis. WSDL is the most important part in a Web Service, which consists of all information needed for the client or requester to decide if Web Service is suitable for his requirements or not. He can address issues such as which Web Service he wants to execute? , how can the requester access this Web Service? What are the parameters that Web Service messages need? And what are the datatypes of these parameters? If the requester wants web service, firstly he examines its WSDL document to determine what functions are available with this service and if it is able to carry out his needs.

Thus, as mentioned, we can see that a WSDL document has many parts which are listed as follows (W3C, 2008):

1. **Messages**: XML-Based format defined as an abstract set of data arranged in message format which implements the data traveling from one endpoint to another by specifying the structure of the input and output messages (W3C, 2008) .

2. **Operations**: XML-Based format defined as naming a method, message queue, or business process that will accept and process the message. It is also defined as the SOAP action and the way the message is encoded (W3C, 2008) .

3. **Port Type**: XML-Based format defined as an abstract set of operations mapped to one or more endpoints, and also defined as description of the interface of Web Service (W3C, 2008) .

4. **Binding**: XML-Based format defined as concrete protocol and data formats for the operations and messages for a particular port type (W3C, 2008).

5. **Port**: XML-Based format defined as a combination of a binding and a network address ,providing the target address of the service communication (W3C, 2008).

6. **Service**: XML-Based format defined as a collection of related end points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions (W3C, 2008) .

7. **Datatypes**: XML-Based format defined as specification of the input and output parameter data type (W3C, 2008) . There are many challenges to implementing all of the data type such as array and array of object. These challenges will be discussed in this research and we will try to solve this problem .

Figure (2-3) shows a sample of WSDL document. The document describes the *HelloService*, the service provides a single publicly available function, called sayHello. The function expects a single string parameter, and returns a single string greeting. For example, if a requester passes the parameter *" world* ", the service returns the greeting, "Hello, world!"

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
   targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>
   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>

   <binding name="Hello_Binding" type="tns:Hello_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="sayHello">
         <soap:operation soapAction="sayHello"/>
         <input>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </input>
         <output>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </output>
      </operation>
   </binding>

   <service name="Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
         <soap:address
            location="http://localhost:8080/soap/servlet/rpcrouter"/>
      </port>
   </service>
</definitions>
```

Figure (2-3): *HelloService.wsdl* document

As we mentioned, WSDL is complex in nature. Even developers themselves acknowledge this complexity is due to the way in which it was written, and it requires using several

different tools such as Java, axis2 etc (R. Grønmo et al, 2004) and (A. Bellini et al,2010) so that the WSDL document may differ from one tool to another.

WSDL documents have full descriptions for Web Service and contain all information about Web Service. A WSDL document can be considered a guide for a service requester to determine if the Web Service is suitable for his needs or not, and also help him to use it. For example, the WSDL document will tell the service requester how to access service, the parameters that he needs to register and bind with Web Service, and also the operations which the Web Service implements (R. Virgilio et al, 2010) and (K. Kumar et al, 2009).

Because the WSDL document contains all this information about Web Service, it must contain huge amount of data. These data must be arranged in a formal way with an orderly sequence of parts. This sequence is called XML-Schema (H. EL Bouhissi et al, 2009) and (E. Domínguez et al, 2007) which will be discussed next.

2.6 Simple Object Access Protocol (SOAP)

The Service-Oriented Architecture (SOA) is defined as the activities and the roles of architectural elements. This architecture is built on a foundation of standards to support the interoperability of heterogeneous applications across the network with respect to remote communication protocol SOAP (Simple Object Access Protocol) (A. Arsanjani et al, 2002).

SOAP is an XML-Based format, defined by several meanings. A general definition (defined by WC3) indicates that SOAP is "a lightweight protocol intended for exchanging structures information in a decentralized, distributed environment," and also defined by (A. Arsanjani et al, 2002) as communication protocol used to call remote operations by exchanging XML-Based messages. SOAP messages can be transported over HTTP for the runtime invocation. The HTTP protocol plays the bridging role for interactions between computer systems (H. Wang et al, 2004).

2.7 Universal Description, Discovery and Integration (UDDI)

UDDI is a platform-independent, (XML)-based registry by which businesses worldwide can list themselves on the Internet (S. Kumar et al, 2012), and a mechanism to register and

locate web service applications. UDDI was originally proposed as a core Web service standard. It is designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with the web services listed in its directory.

We explained all Web Service components and also mentioned all the properties of each part separately. But the essence of this research focuses on WSDL, and we will illustrate the main research problem in XML-Schema in next section.

2.8 XML-Schema

XML is W3C recommendation which was designed mainly to transport and store data but not to display it (W3C, 2008). It was also created to structure information and services. Really, it is an intermediary tool that handles the differences in data types between Web Services. It is important to say that W3C supported an XML-Based alternative to DTD (Data Type Definition) called XML-Schema. This schema is not predefined but the developer must define his own tags; it is also designed to be self-descriptive.

XML-Schema is more implementation-oriented and includes programmatic and syntactic detail (E. Dominguez et al, 2008). As we mentioned earlier, user-defined data types and compositions are not translated in the same way, since XML schema does not support all datatypes. Transporting data in XML often requires a higher degree of type checking to ensure robustness in document understanding and data interchange (W3C, 2008). Object-oriented type systems have several limitations and problems in representing the XSD constraints. Some of these problems are due to the mismatch of XML-Schema and Object Oriented type system. These limitations are listed bellow according to (S. Alagic et al, 2010):

1. Not distinguishing between elements and attributes in the object-oriented representation or not representing attributes at all.
2. Not being able to represent repetition of elements and attributes with identical names (tags).
3. Failing to represent correctly the particle structure of XSD (with elements and groups) and the range of occurrences constraints in particular.

4. Confusing the particle hierarchy (with elements and groups) and the type hierarchy (with simple and complex types and type derivations) of XSD.
5. Not distinguishing different types of XSD groups in the object-oriented representation (sequence versus choice) or not representing groups at all.
6. In the object-oriented representation, not distinguishing the two type derivation techniques in XSD: by restriction and by extension.
7. Failing to represent accurately XSD type derivation by restriction, facets and range constraints in particular.
8. Having no representation of the XSD identity constraints (keys and referential integrity) and thus no way of enforcing them.

As a result, for those issues and challenges, we note that XML-Schema do not support all data types and also object-oriented type systems. Therefore, the main contribution of this thesis is attempting to extend XML-Schema Data types to get clear description for Web Information Systems that will allow any non-expert Web service requester to understand the method signature and data structure of Web service.

2.9 XML-Schema Datatypes

XML-Schema is a way to specify the legal or acceptable building block of an XML document. XML-Schema is used to put constraints on the elements and attributes that can be in an XML document instance, used to define the relations between the elements, and also to define the datatypes associated with the elements and attributes.

According to (W3C, 2008) , XML-Schema datatype can be categorized into two types:

1. Simple datatypes:
   Simple datatype include (S. Hanna et al, 2011) :
   a. Built-in primitive datatype for example *string*.
   b. Derived from built-in primitive datatype: these datatypes are derived from built-in primitive datatype by applying some default constraints, for example *nonPositiveInteger* is derived from *integer* by restricting the value space of integer to only negative number .

c. User-derived datatypes: User-derived datatypes are simple datatypes derived by restricting a base datatype (which can be a built-in primitive or derived from primitive datatype) using constraining facets such the *enumeration* constraining facet .
d. List datatypes: consist of a finite length sequence of values of built-in, derived from built-in or user derived datatypes. All the values of a list need to have the same datatype.
e. Union datatypes: the union of the values of one or more datatypes.

2. Complex Datatypes:
   Complex datatypes consist of one or more elements and attributes of the datatypes (S. Hanna et al, 2010).

2.10 Constraints Modification

This part of WSDL document refers to the constraints which added to the input and output parameters elements. This part of the WSDL document contains a provision for specifying the occurrence or how many specifying elements will appear.

2.11 Summary

In this chapter, we have introduced a background about Web Services components and Web services roles; we also discussed the XML-Based and XML-Schema. We mentioned the problems with Web Services (S. Hanna, 2008) and the main problems with XML-Schema according to (S. Alagic et al, 2010). We are interested in datatypes problems which are not represented in a clear way using XML-Schema, so the main contribution of this thesis are to extend the XML-Schema in order to overcome these problems.

CHAPTER THREE

LITERATURE REVIEW FOR WEB SERVICE UNDERSTANDABILITY

As it is well known Web Services include a large number of research fields, many studies and researches have been published in the Web Service area. For example, if we take a sample of these studies, we note that some of the researches focused attention on how to build a Web Service. Other researchers proposed approaches for specifying semantic Web Services composition using UML (Unified Modeling Language) profile (J. Timm et al, 2007) and (E. Domínguez et al, 2007). Other researchers recommend a model-driven process for web services development (R. Grønmo et al, 2004), and there are many others.

3.1 Overview

Many recent studies have been published in the field of Web Service. The goal of these studies is to facilitate and increase the communication among the distributed systems, and also use Web Service reverse engineering to facilitate the reuse and composition of the Web Services (W. Sun et al, 2009), to ultimately facilitate the exchange of information over the networks. In order to facilitate the understandability of the Web Services functionality and what these Services are offering to its requester (E. Domínguez et al, 2007), a way must be found to facilitate the description of Web Services.

While reviewing several previous studies concerning the field of Web Services, obviously it was necessary for all researchers to mention several main concepts, such as XML, UML, SOAP, XSD and also WSDL (A. Bellini et al, 2010), which in turn are used to perform selections, descriptions, discovery, composition, and interaction with the Web Services (E. Sirin et al, 2004). All researches attempted to built a bridge between the Web Services providers and Web Services requesters to reach better comprehension for Web Service functionality from the Web Service requester to increase the exchangeability of information between heterogeneous applications .

The related research to this thesis is about representing the information inside WSDL in a more understandable form .

In this thesis, we classify the related Web Service understandability into several aspects and we also give a brief overview for each research and the limitations for each research which we will try to solve in this thesis.

3.2 Reverse Engineering Approach for Semantic Web Services Composition

(W. Sun et al, 2009), presented an approach to facilitate and raise the degree of automation for Composition of Web Services. The approach used UML-Model to give graphical description for the Composition Web Services, The proposed approach summarized in three steps of Web Services Composition:

1. Using RE methodology to turn the selected Web Services WSDL documents to one or more UML-Model depending on the number of selected services.
2. Integrating these models into one UML-Model which implements all integrated UMLs using one of the UML-tools.
3. In step 2 a new Web Service is created (Composition Web Service) and by using one of the UML-tool a new description for this Web Service is created which called OWL (Web Ontology Language) .

Here we can criticize this work in simple terms. The final description OWL is dependent on UML-Models. These models are created using different tools and also may implement heterogeneous applications. Suppose one or more of these applications is used by one of the Datatypes not implemented clearly in WSDL, such as char, array, array of objects. Here, the model which implements the Web Service before composition will have ambiguity, but after it composes with others, inevitably the ambiguity will increase, so that this approach is good and will work properly if all of the datatypes of Web Services parameters are represented clearly. If one or more parameters are represented ambiguously, surely it will face missed understanding for Web Services requesters and developers. Our proposed approach seeks to overcome these challenges and also to reach batter comprehension for Web Service functionality.

3.3 Model-Driven Web Service Development

In this field (R. Grønmo et al, 2004) has been proposed another way to give more comprehension for the Web Service description to make it easy for a requester to decide if the selected Web Service is applicable for his requirement or not.

The proposed approach summarized in three steps of Model-Driven Web Service Development which divided into following steps:

1. The WSDL are converted to graphical modeling language (UML) .

2. Integrate with other UMLs for a composition Web Service.
3. A new Web Service descriptions are exported.

This approach is not different from the previous one but it added a Pure UML modeling strategy supported by implementation of two-way conversion rules from WSDL to UML and also from UML to WSDL documents. But this rule does not avoid the issues and problems which we are trying to solve, so the same challenges of understandability are still present.

3.4 Reverse Engineering Existing Web Service Applications

(H. EL Bouhissi et al, 2009) proposed a novel approach based on reverse engineering. This approach is split in two stages: reverse engineering to extract the useful information from the WSDL document, and engineering for constructer of the Web Service. The proposed approach is adding a semantic to the Web Service according to Web Service Modeling Ontology, to facilitate for the Web Service clients to discovery, selection, composition, and also execution of the Web Service.

A reverse engineering approach reduces the effort and cost to build a new Semantic Web Service by adding a semantic layer to an existing Web Service using a description file WSDL without referring to source code. In this case the semantic for Web Services will be built depending on WSDL. As we mentioned earlier, the semantic for Web Service will suffer with the same problems because the WSDL document may contain one or more aforementioned dtatatypes, and that will lead to a misleading semantic for a Web Service, resulting in the Web Service not having a good route for selection, composition, discovery from the users because a user cannot understand the functionality of the Web Service to decide if it is applicable for his purposes.

3.5 Meta-Modeling of Semantic Web Services

This research discussed other manners for dealing with the understandability of the Web Service, called Meta-Model. This manner is proposed by (R. Virgilio et al, 2010) , and it allows interoperability at different levels of abstraction.

The Meta-Model Approach is summarized according to (R. Virgilio et al, 2010) in three levels: a conceptual level, a logical level and a physical level which are illustrated as follows:

1. A conceptual level, proposing a simple conceptual model where a set of constructs properly represents semantic concepts. Each construct is used to properly represent elements of documents, with the same semantics.
2. A logical level, implementing the conceptual model into a logical one. In this case they used the relational model.
3. A physical level, defining the physical design of the logical representation of previous level.

This approach is different from others in the literature, as it provides implementation solution starting with the definition of a meta-representation of the chosen data model at a conceptual level. The main challenges which we attempt to solve are not exceeded and also we must note that the understanding of Web Service functionality by its users depends on the parameters datatypes which are used to implement the Web Service operations. This approach is not effective if the Web Service used one or more of the aforementioned dtatatypes, as that will lead to a misleading comprehension for the Web Service that should be avoided.

3.6 Top-Down Approach for Web Service Development

This approach performs the analysis of the processes before the services development. (A. Bellini et al, 2010) Proposed top-down development approach considered as the most efficient mechanism to build new Web Services. In this approach, the WSDL file allows one a greater control over the Web Services and it can eliminate interoperability problems that may arise during the creation of a Web Services using the bottom-up . This approach

as all proposed approaches dealing with WSDL documents without any care about the mismatching of the parameters datatypes so it will suffer the same problems of ambiguity to select, reverse engineering, reused the Web Services.

3.7 WSDL automatic generation from UML model in MDA framework

One of other approaches which proposed to deal with Web Services description is MIDAS-CASE. It aimed to extend the UML language to support the modeling of the Web Services description, and then automatic generation of the WSDL document for concerned Web Service (J. Vara et al, 2005).

As we illustrated before, a WSDL file is an independent XML-based standard which is proposed by W3C to represent the Web Services functionality (J. Vara et al, 2005). One of the other properties for WSDL is that it is XML-based version of Interface Definition Language (IDL), so MIDAS-CASE is one of the framework methodologies that aimed to facilitate the development of Web Service Information System depending on Model Driven Architecture (MDA). MDA has three dimensions: CIM (Computation Independent Model), PIM (CIM Platform Independent Model), and finally PSM (Platform Specific Model), (J. Vara et al, 2005) used in his approach.

This approach is divided into three steps:

1. The client defines extended UML model which is then stored as XML-based.
2. (J. Vara et al, 2005) defined an XML-Schema to describe Meta-Model for extended UML which was introduced in step one, and then WSDL document is automatic generated using one of the existing tools.
3. The XML document now becomes an instance of the XML-Schema. The XML document is not valid according to the XML-Schema, thus we conclude that the model is not valid, since the model does not carry out the metamodel.

This MIDAS-CASE web service architecture is one of the most important approaches used for a Web Service development, but as shown it doesn't deal with WSDL document which are automatically generated and also built according for extended UML model. UML models introduced and extended are vulnerable for the risks and challenges previously mentioned and never get clear understandability for Web Services functionality.

3.8 Summary

In this chapter we discussed recent samples of research which aims for understanding Web Service functionality. As shown, the proposed approaches use UML models to express the Web Service description, but they have ignored dealing with the operation input/output parameters datatypes. However the Web service description remains unclear because the description of parameters datatypes is differently expressed from tool to another, so the UML description will be different for the same Web Service if programmed at different tools.

CHAPTER FOUR

THE PROPOSED MODEL: EXTENDING THE XML-SCHEMA DATATYPES SPECIFICATION TO REACH A BETTER COMPREHENSION OF THE WSDL DOCUMENTS

Based on the previous review of Web Service understanding we noted that no approach attempted to solve the inconsistency and ambiguity in defining the Web Service operations parameters datatypes.

4.1 Datatypes Description

The previous approaches solved the problems of the Web Services understanding, reusing and comprehension by using UML to give graphical definitions for Web Services and its functionality. These approaches have several advantages such as :

1. The graphical implementation gives a full summary for the Web Service functionality but with few details.
2. The graphical implementation is easier to understand than textual implementation (WSDL document).
3. Can be easily understood even by non-specialists in Web services.

However the graphical implementation ignores the most important part which is the needed data that must be used to bind with Web Services, on which we are focusing in this thesis.

As motioned in chapter two, messages are used to bind with Web Services by filing an application which published by the Web Service provider with parameters. Surely these parameters must clearly appear to the users without ambiguity; because any error in the filling of these parameters will lead to Web Service failure which we always seek to ensure does not happen. Therefore we are proposing an approach based on extending the XML-Schema to reach better comprehension and reusing the Web Services functionality, which in turn leads users to understand all Web Services operations and also to determine all the parameters datatypes which Web Services need.

The proposed approach is accompanied with a tool in order to prove the approaches usefulness and compare it with other approaches. The tool can be auto run when the user tries to bind with the Web Service. This tool can answer the major question of this thesis, that is: Can we extend the XML Schema datatypes to reach for a better comprehension of

the WSDL documents by the service requester and provider of Web Services? Other questions could be inspired from the previous major question, such as:

1. Do all of the parameters datatypes need to be extended ?
2. Can the tool distinguish between the parameters datatypes ?
3. How can we reach better comprehension for the Web Services ?

Section 4.2 shows the model which this thesis proposes to solves the datatypes description problems, and we used several datatypes as case studies to illustrate all model steps. These datatypes can be divided into three categories:

1. Clear Datatypes .
2. Indistinguishable Datatypes .
3. Unclear Datatypes .

Figure (4-1) shows how the tool deals with these datatypes.

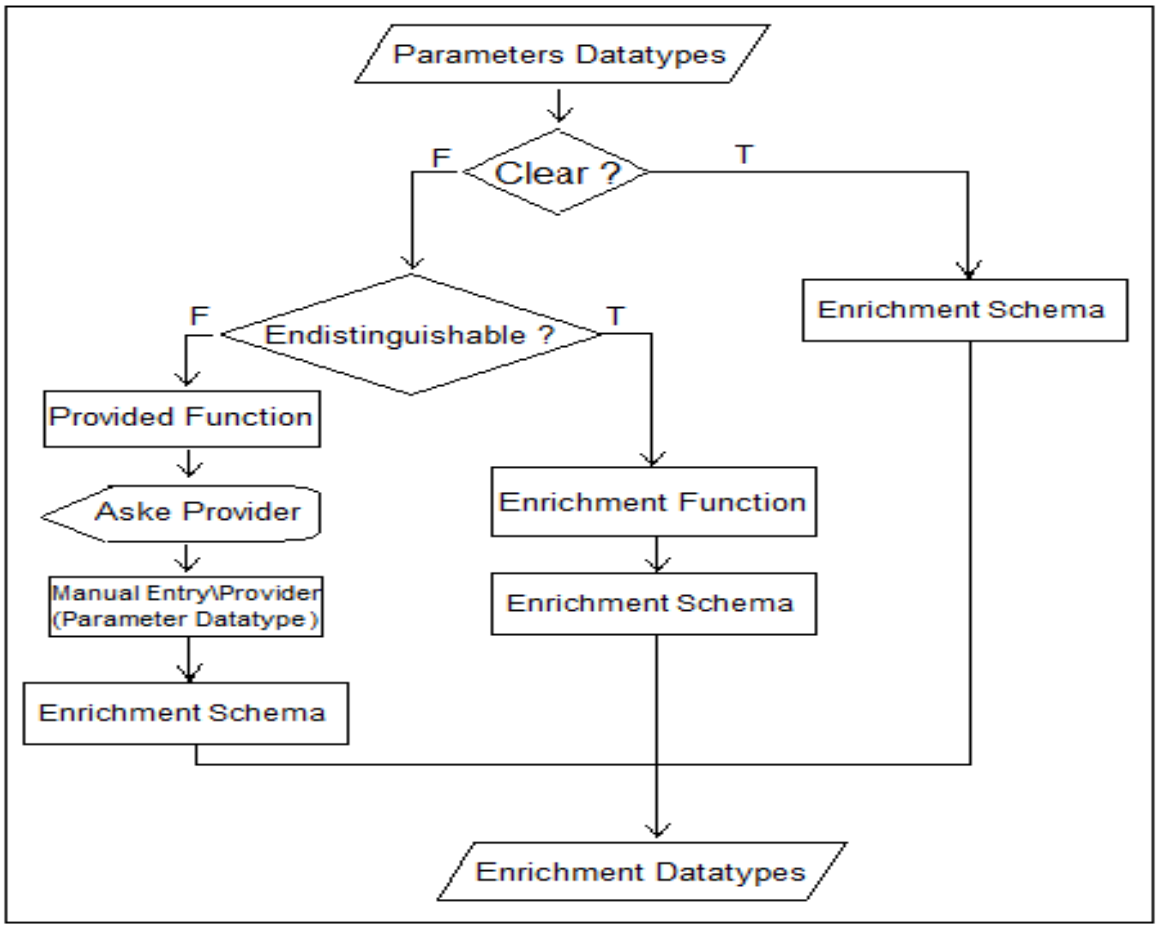


Figure (4-1): The datatypes processing

4.2 The Proposed Model

The proposed model consists of five phases: a) extract the WSDL document, b) extract datatype specification, c) add more description and annotation, d) add constraining facets to the datatypes, and e) extract UML class diagram using any published tool.

Figure (4-2) shows the general structure of the proposed model.

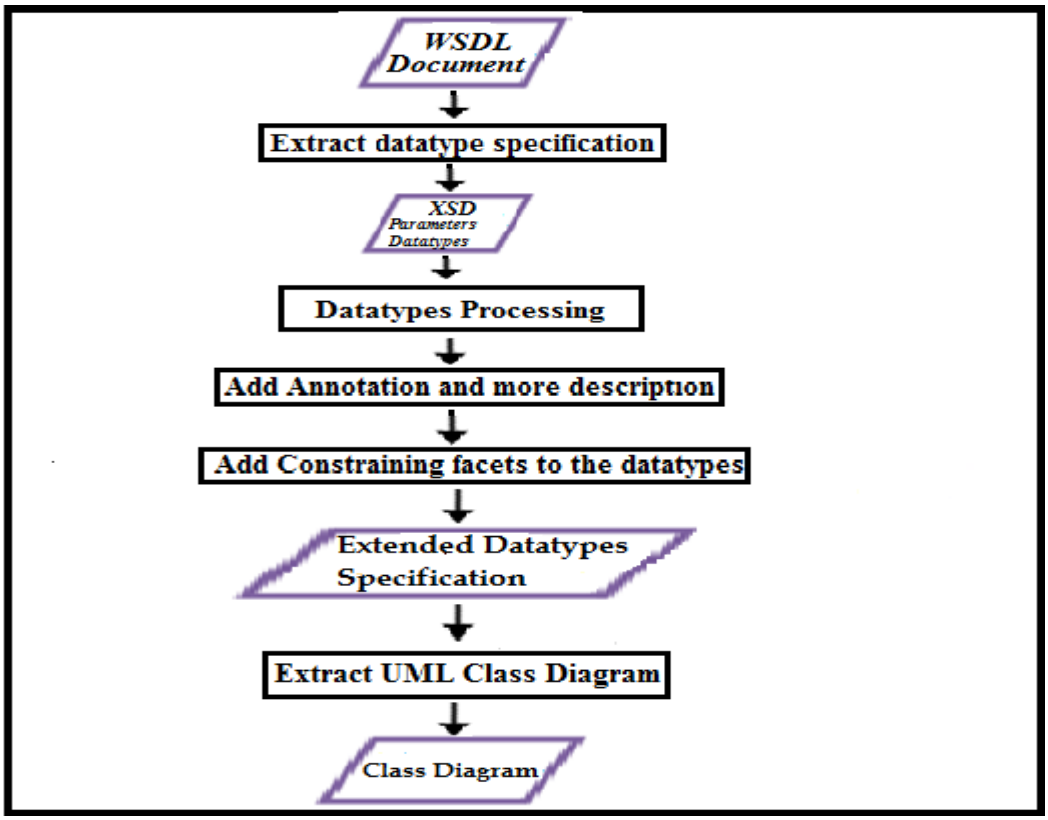


Figure (4-2) : The proposed model

The input of proposed approach is a WSDL document. WSDL description document is complex in nature, usually automatically generated by one of the Web Services development tools such as .NET, Apache Axis, Java etc. (R. Grønmo et al, 2004). Each of these tools has its own particular way to define or to implement the input and output parameters datatypes for each of the Web Services operations. Here we are attempting to overcome these differences.

The first step addresses the question of how to extract the WSDL document. WSDL documents are compulsorily published with Web Service; the provider cannot publish his own service application until its description (WSDL) generated, so that any developer or user wanting to know more about the operations or services then he can review the provided WSDL document.

There are many ways to extract WSDL document, but here we are looking to make our proposed tool to run automatically when the Web Service client, user, and also developer want to bind with the Web Service and in the final stage give him a clear and simple

description for Web Service input/output parameters datatypes. The proposed tool extracts the WSDL document and then extracts the XSD. Then the tool can distinguish between the input/output parameters datatypes which may need more description and constraints with which do not need.

First we discuss how different .NET , Java Datatypes are specified inside WSDL, given that WSDL documents depends on the XML Schema Data types (XSD) system, datatype specification produced when using an Axis2 based tool to build a Web Services is also compared.

Suppose we have a method with *byte* and *char* Datatypes parameters, the question here is: how do these parameters will be implemented inside WSDL using the aforementioned platforms ?. Next examples will illustrate that.

1- byte Data Type

a) The *byte* in C# :

Table (4-1): *byte* data type and its alias (Byte)

Short Name	C# Class	Type	Width (bits)	Range
Byte	Byte	Unsigned integer	8	0 to 255

XSD Equivalence of C# *byte* :

byte or its alias *Byte* are equivalent to *unsignedByte* in XSD

Example 1

For the following method

```
byte byteExample(Byte bytepar)
```

The input and output parameters datatypes of previous method are specified by XSD inside WSDL as Figure (4-3).

```
<xs:element name="byteExample">
<xs:complexType>
<xs:sequence>
<xs:element type="xs:unsignedByte" name="bytepar" >
minOccurs="0"/>
</xs:sequence>

</xs:complexType>
</xs:element>
<xs:element name="byteExampleResponse">
<xs:complexType>
<xs:sequence>
<xs:element type="xs:unsignedByte"
name="byteExampleResult" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Figure (4-3): WSDL document (*byte* datatype using .NET)

The *byte* datatype is defined using .NET as *unsignedByte* for each operations request and response .

a. <xs:element type="xs:*unsignedByte*" name="bytepar" >.
b. <xs:element type="xs:*unsignedByte*" name="byteExampleResult">.

b) *byte* in Java :

Table (4-2): *byte* data type and its Wrapper (*Byte*)

Name	Wrapper Class	Type	Width (bits)	Range
byte	Byte	Signed integer	8	-128 to 127

XSD Equivalence of Java *byte*

byte or its Wrapper class *Byte* are equivalent to *byte* in XSD.

Example 2

For the following method

```
byte byteExample(Byte bytepar)
```

The input and output datatypes for above method are specified by XSD inside WSDL as Figure (4-4).

```
<xs:complexType name="byteExample">
 <xs:sequence>
    <xs:element type="xs:byte" name="arg0" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="byteExampleResponse">
<xs:sequence>
    <xs:element type="xs:byte" name="return"/>
</xs:sequence>
</xs:complexType>
```

Figure (4-4): WSDL document (*byte* datatype using Java)

The *byte* datatype is defined using java as *byte* for each operations request and response without any changes :

a. <xs:element type="xs:*byte*" name="arg0" minOccurs="0"/>.
b. <xs:element type="xs:*byte*" name="return"/>.

c) Axis2 Equivalence of *byte*

Example 3

For the following method

```
byte byteExample(Byte bytepar).
```

The input and output datatypes are specified by XSD inside WSDL as Figure (4-5).

```
<element name="byteExample">
        <complexType>
          <sequence>
            <element name="bytepar" type="xsd:byte"/>
          </sequence>
        </complexType>
      </element>
      <element name="byteExampleResponse">
        <complexType>
          <sequence>
            <element name="byteExampleReturn" type="xsd:byte"/>
          </sequence>
        </complexType>
      </element>
```

Figure (4-5): WSDL document (*byte* datatype using Axis2)

The *byte* datatype is defined using Axis2 as *byte* for each operations request and response with no changes :

a. `<element name="bytePar"type="xsd:byte"/>`

b. `<element name="byteExampleReturn" type="xsd:byte"/>`

2- char datatype

a) *char* in .NET

Table (4-3): *char* data type and its alias

Short Name	.NET Class	Type	Width (bits)	Range
char	Char	A single Unicode character	16	Unicode symbols used in text

XSD Equivalence of .NET *char*

The WSDL document for the Web Service which used *char* or *Char* alias datatype as request and response operations is defined as custom datatype (ns : char) where (ns) is a .NET namespace. *char* or its alias *Char* defined inside WSDL as the following example:

Example 4

For the following method

```
public char charExample(Char charPar)
```

The *char* datatypes are specified by XSD inside WSDL as Figure (4-6).

```
<xs:element name="charExample">
   <xs:complexType>
<xs:sequence> .
<xs:element xmlns:q1="http://schemas.microsoft.com/2003/10/
Seria
lization/" minOccurs="0" name="charInput" type="q1:char"/>
  </xs:sequence>
  </xs:complexType>
     </xs:element>
       <xs:element name="charExampleResponse">
  <xs:complexType>
    <xs:sequence>
<xs:element xmlns:q2="http://schemas.microsoft.com/2003/10/Serializ
ation/" minOccurs="0" name="charExampleResult" type="q
2:char"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Figure (4-6): WSDL document (*char* datatype using .NET)

The *char* datatype is defined using .NET as *q1:char* for each operations request and response with .NET namespace:

a. `<xs:element xmlns:q1="http://schemas.microsoft.com/2003/10/Serialization/" minOccurs="0" name="charInput" type="q1:char"/>.`

```
b. <xs:element xmlns:q2="http://schemas.microsoft.com/2003/10
   /Serialization/" minOccurs="0" name="charExampleResult" ty
   pe="q2:char"/>
```

b) *char* in Java

Table (4-4): *char* data type and its alias (Byte)

Name	Wrapper Class	Type	Width (bits)	Range
char	Character	A single Unicode character	16	'\u0000' (or 0) to '\uffff' (or 65,535)

XSD Equivalence of Java *char*

char or its Wrapper class *Character* are equivalent to *unsignedShort* in XSD

Example 5

For the following method

```
char charExample(Character charPar)
```

The char datatypes for previous method are specified by XSD inside WSDL as Figure (4-7).

```
<xs:complexType name="charExample">
     <xs:sequence>
<xs:element name="arg0" type="xs:unsignedShort" minOccu
rs="0"/>
     </xs:sequence>
    </xs:complexType>
    <xs:complexType name="charExampleResponse">
    <xs:sequence>

<xs:element name="return" type="xs:unsignedShort"/>
       </xs:sequence>
    </xs:complexType> .
```

Figure (4-7): WSDL document (*char* datatype using Java)

The *char* datatype is defined using java as *unsignedShort* for each operations request and response:

```
a. <xs:element name="arg0"type="xs:unsignedShort"minOccurs="0"/>.
```

```
b. <xs:element name="return" type="xs:unsignedShort"/> .
```

c) Axis2

For the following method

```
char charExample(Character charPar)
```

The following warning was generated

The service class "wtp.Datatypes" does not comply with one or more requirements of the JAX-RPC 1.1 specification, and may not deploy or function correctly.

The method "charExample" on the service class "wtp.Datatypes" uses a data type, "char," that is not supported by the JAX-RPC specification. Instances of the type may not serialize or deserialize correctly. Loss of data or complete failure of the Web service may result, and the following datatype specification was generated inside WSDL.

```
<element name="charExample">
    <complexType>
     <sequence>
      <element name="charPar" type="xsd:anyType"/>
     </sequence>
    </complexType>
   </element>.
```

In this section, we illustrate the implementation differences between three tools, and we also show how each of these tools implement *byte, char* input output parameter datatypes. These differences create misunderstandings for the Web Services requesters, clients, users and also developers because these datatypes are not implemented in the same and formal way as we have seen in *char* datatype. But here in our thesis we want to implement our proposed approach on .NET tool as case study.

4.3 Constraints Modification

In all of the previous examples we provided an illustration for parameters datatypes but other aspect will process by our proposed tool which is the constraints for these parameters, so in next paragraphs we illustrate this aspect.

We noted in previous figures, WSDL contain the **< xs:element .. minOccurs="0" name ...>** part, it contains ***minOccur="0"*** and also in other case may contain ***maxOccur =" "***.

To illustrate this aspect we discussed some possible case studies. Figure (4-8) includes three examples of occurrences of a specific element.

```
1. <element name="one" type="string" minOccurs="3" maxOccurs="4"/>.
2. <element ref="target:one" maxOccurs="10"/>.
3. <element name="position" ""minOccurs="0" maxOccurs="unbounded"/>.
```

Figure (4-8): min/max occurrence cases

In example 1 Figure 4.8, it declares that element **<one>** should appear within the instance document a minimum of three time and a maximum of four times. Example 2 declares an element using a reference to global **<one>** declaration with maximum attribute with 10 time appearance. The last example specifies the element **<position>** which may not appear at all **"minOccurs="0"** and it may also appear for infinite number of times **maxOccurs="unbounded**" . The default value for each minimum and maximum is 1 time appearance, meaning if not specified by provider, then the element must be appear for one time at least. An additional constraint to which the provider must adhere when specifying min and max occurrence is that the max value must always be greater than or equal to the min value.

4.4 Proposed Tool Environment

The tool solves the understandability problem by the creation of new XML-Schema called enrichment schema. This schema, simplified as much as possible, consists of just the WSDL parts which the requesters need to know what the concerned Web Service serve. Chapter five consists of the enrichment schema, the enrichment algorithm, interfaces for how our proposed tool runs, and also a table for 33 data types with examples for each type and the method to implement inside WSDL documents and how these data types are classified by our proposed tool.

4.5 Datatypes Classifications

In this section we show how the proposed tool can be distinguished between the datatypes and how it deals with these differences. In Figure 4.1 we show three groups for parameters datatypes; these types are included in three possible cases.

4.5.1 Clear Datatypes :

In this case, the datatypes are implemented in a formal way and the datatypes are implemented as it is without any changes, so there is no need for any enrichment. The new WSDL document generated by our proposed tool (Extended XML-Schema) will have the same XSD datatypes without any modification to the original WSDL document. The enrichment part will have the same implementation for the datatypes with no changes, as the datatypes are clear and need no annotations. We will show the enrichment schema in chapter five with more details. The next example shows how the .NET tool implements *double* datatypes as case study and also shows how the proposed tool deals with this case.

***double* datatype**

1- *double* in C#

Table (4-5) : *double* data type

Short Name	.NET Class	Type	Width (bits)	Range
double	Double	Double-precision floating point type	64	-1.79769313486232e308 to 1.79769313486232e308

XSD Equivalence of C# *double*

double or its alias *Double* are equivalent to ***double*** in XSD.

Example 6

For the following method

```
public double doubleExample(Double doublePar)
```

The *double* datatypes are specified by XSD inside WSDL as Figure (4-9).

```
<xs:element name="doubleExample">
  <xs:complexType>
    <xs:sequence>
<xs:element minOccurs="1" name="doublePar" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="doubleExampleResponse">
  <xs:complexType>
   <xs:sequence>
<xs:element minOccurs="1" name="doubleExampleResult" type="xs:double"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure (4-9): WSDL document (*double* datatype using .NET)

This example for double datatype implementation shows how C# tool implements the double datatype inside WSDL document:

a. **<xs:element minOccurs="0" name="doublePar" type="xs:*double*"/>**.

b. **<xs:element minOccurs="0" name="doubleExampleResult" type="xs:*double*"/>**

The proposed tool will firstly extract the WSDL document and then extract the XSD part, and finally check if the datatype is clear or not. In this example the tool will skip the third and forth steps of our proposed model because there is no need for any annotations or constraints. The parameter (*doublePar*) is given its type *double* without any ambiguity. The following steps summarize how the tool functions:

1. Step 1:

   Extract the WSDL document for the `(public` ***double*** `doubleExample(`***Double*** `doublePar))` method ,which shown in Figure 4.9 .

2. Step 2:

   Extract the parameter datatypes XSD as:

   a. **<xs:element ... type="xs:*double*"/>** **(**Input parameter).

b. `<xs:element ... type="xs:double"/>` (Output parameter).

In this phase the tool can be distinguished that these parameters is not needs for more description it is clear and the requester can know that it is *double* datatype as it is.

3. Step 3:

No annotation to be added. The enrichment part will have the same implementation for datatype as it is in original WSDL document with no annotations.

4. Step 4:

Add constraints that the elements *"doublePar"* and *"doubleExampleResult"* will appear one time or more for both as :

<enr_min_appear> "1" </enr_min_appear>

<enr_max_appear> "unbounded" </enr_max_appear>

5. Step 5:

The class diagram will not be changed after we run our proposed tool because the WSDL document has not changed. We will show how the proposed tool introduced the enrichment schema and also the enrichment algorithm for the three classification datatypes in chapter five.

4.5.2 Indistinguishable datatypes

Here other cases of datatypes are discussed. In example 5, we shows one of the datatypes which is clearly implemented inside WSDL document, but here we will show other cases of the datatype (class datatype as a case study).

The proposed tool can distinguish the mismatch defined, the WSDL document extracting then XSD extracting and then apply the third and fourth steps by adding more descriptions (annotations, constraints).

***Classes* datatype**

1- *Classes* In C#

This sample of method code shows the implementation for player member which defined as class with two parameters, his name and his nickname both of parameters defined as *string* datatype.

```
[DataContract]
    public class Player
    {
        [DataMember]
        public String Name1 { get; set; }

        [DataMember]
        public String NickName {get; set;}
      }
```

The *string* datatypes are specified by XSD inside WSDL as Figure (4-10).

```
http://localhost:50571/Service1.svc?xsd=xsd2
        <xs:sequence>
        <xs:element minOccurs="0" name="Name1" nil
        lable="true"    type="xs:string"/>
        <xs:element minOccurs="0" name="NickName"
        nillable="true" type="xs:string"/>
        </xs:sequence>
        </xs:complexType>
        <xs:element name="Player" nillable="true"
        type="tns:Player"/>
```

Figure (4-10): WSDL document (*class* datatype using .NET)

This example for classes applied to one of the datatypes that can be distinguished by the proposed tool. The class here (`<xs:element name="Player" nillable="true" type="tns:Player"/>`) has two parameters and in other case may have more. Each of these parameters is defined in a separate line so that the proposed tool can determine that these parameters belong to the class datatype. The annotations and constraints will be added to the new enrichment XML-Schema to give more description than the original WSDL document.

Now we can show how our proposed tool deals with *uint* datatype .The *uint* is presented as following example in Figure (4-11).

```
public uint uintExample(UInt32 uintpar)
```

Figure (4-11): uint datatype

The WSDL document for this method show in Figure (4-12) .

```
<xs:element name="uintExample">
  <xs:complexType>
    <xs:sequence>

<xs:element minOccurs="0" name="uintPar" type="xs:unsignedInt"/>

    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="uintExampleResponse">
  <xs:complexType>
     <xs:sequence>

<xs:element minOccurs="0" name="uintExampleResult" type="xs:unsignedInt"/>

     </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure (4-12): WSDL document (*uint* datatype using .NET)

The uint datatypes is implemented inside WSDL for (Figure 4.11) method as *unsignedint* which custom declaration for .NET, and it may be represented using other tool by other way so this datatype process by our proposed tool as following.

1. Step 1:

   Extract the WSDL document for Web Service. The previous example illustrates this step in figure 4.12.

2. Step 2:

   Extract the parameter datatypes XSD as:

   a. `<xs:element ... type="xs:unsignedInt"/>` (Input parameter).

   b. `<xs:element .. . type="xs:unsignedInt"/>` (Output parameter).

3. Step 3:

   Add annotation for the proposed schema that provides the correct type for the input parameter " *uintPar* " and output parameter *"uintExampleResult"* is *uint* type for both as :

   `<enr_type > "uint" </enr_type>`

4. Step 4:

   Add constraints that the input parameter *" uintPar "* and output parameter *" uintExampleResult "* will appear zero times or more for both as :

   `<enr_min_appear> "0" </enr_min_appear>`

   `<enr_max_appear> "unbounded" </enr_max_appear>.`

5. Step 5:

   Class diagram will be generated during any published tool depending on the new enrichment WSDL document.

4.5.3 Unclear Datatypes:

Here is the third classified datatype which cannot be addressed until back to the Web Service provider itself. The tool can execute step 1 and step 2 and then checking about the datatype classification. In the previous two classifications the tool can address the problem automatically; in unclear datatypes it stops and asks the Web Service provider about which datatypes the provider specified for Web Service operation parameter datatypes.

We discussed this case using Array and List as case study

Array and List Datatype:

Suppose the provider defined the following sample code of the Web Service code in Figure (4-13).

```
int[] ArrayExample(int[] arrayPar.
```

Figure (4-13): *Array of int* Datatype

The WSDL document for previous declaration of array show in figure (4-14).

```
<xs:complexType>
<xs:sequence>
<xs:element xmlns:q3="http://schemas.microsoft.com/2003/1

0/Serialization/Arrays" minOccurs="0" name="arrayPar" nil
lable="true" type="q3:ArrayOfint"/>

</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="ArrayExampleResponse">
<xs:complexType>
<xs:sequence>
<xs:element xmlns:q4="http://schemas.microsoft.com/2003/1

0/Serialization/Arrays" minOccurs="0" name="ArrayExampleR
esult" nillable="true" type="q4:ArrayOfint"/>

</xs:sequence>
</xs:complexType>
</xs:element>
```

Figure (4-14): WSDL document (*Array of int* datatype using .NET)

The two operations request and response for previous *array* of *integer* declaration are defined inside WSDL as :

a. `0/Serialization/Arrays" minOccurs="0" name="arrayPar" nillable="true" type="q3:ArrayOfint"/>`

b. `0/Serialization/Arrays" minOccurs="0" name="ArrayExampleResult" nillable="true" type="q4:ArrayOfint"/>`

On the other hand (*list* datatype) defined as Figure (4-15).

```
List<int> ListExample(List<String> listPar
```

Figure (4-15): *List of string* Datatype

The WSDL document declaration for the *(list* of *int*) and *(list* of *string*) is shown in Figure (4-16).

```
<xs:element name="ListExample">
<xs:complexType>
<xs:sequence>
<xs:element xmlns:q5="http://schemas.microsoft.com/2003/1

0/Serialization/Arrays" minOccurs="0" name="listPar" nill
able="true" type="q5:ArrayOfstring"/>

</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="ListExampleResponse">
<xs:complexType>
<xs:sequence>
<xs:element xmlns:q6="http://schemas.microsoft.com/2003/1

0/Serialization/Arrays" minOccurs="0" name="ListExampleRe
sult" nillable="true" type="q6:ArrayOfint"/>

</xs:sequence>
</xs:complexType>
</xs:element>
```

Figure (4-16): WSDL document (*List of int* datatype using .NET)

The two *List* of *int* and *list* of *string* are defined inside WSDL as :

a. `0/Serialization/Arrays" minOccurs="0" name="listPar" nillable ="true" type="q5:ArrayOfstring"/>`

b. `/Serialization/Arrays" minOccurs="0" name="ListExampleResult" nillable="true" type="q6:ArrayOfint"/`

Both of *list* and *array* are defined as the same way (`ArrayOfint`, `ArrayOfstring`), both of them are defined as array datatype. The question here is how may the user understand which type of data the operations needs, and how can the user distinguish between the array datatype and list datatype? So that the proposed model can answer these questions by referring to the service provider itself to determine the specific datatype, and then presenting it for a requester in a simple and clear way, the proposed tool functions for this case as follows

1. Step 1:

   Extract the WSDL document for Web Service. The example in Figure 4.16 illustrates this step.

2. Step 2:

   Extract the parameters datatypes XSD as:

   a. *<Serialization/Arrays... "q5:ArrayOfstring"/>*
   b. *<Serialization/Arrays... q6:ArrayOfint "/>*

3. Step 3:

   Here the tool will back to Web Service provider by sending to him an message as interface, asking him to select from a datatypes list which datatype he given for the operation which written its name in the interface. After the provider select the parameter datatype then the tool can add the selected parameter datatype to the enrichment schema .

4. Step4:

   Add constraints that provide how many time the input and output parameters will be appear as we presented before :

   a. <enr_min_appear> " " </enr_min_appear>
   b. <enr_max_appear> " " </enr_max_appear>.

5. Step 5:

   The new class diagram for enrichment schema will created the simplest and clearest way.

In this section we provided three classifications for datatypes which our proposed tool deals with, and we presented an example for each one as a case study, showing how our proposed tool deals with these cases.

4.6 Summary

In this chapter we have introduced an extending XML-Schema based on simplification of the WSDL document for a Web Service. This extending has been applied on the parts which the Web Service requester needs to understand, reuse, select and also reverse engineering of the Web Service. Further, three parts of the WSDL document are required to implement our proposed tool: input/output Parameters Names, input/output Parameters Datatype, and also input/output Parameters Min/Max occurrence. The final enrichment WSDL document gives the simplest and clearest description for the Web Service because the original WSDL document is complex in nature and has a large number of data that are not needed to understand its functionality, so that it not be effective to help the user to decide if it is applicable for his needs or not. But the proposed tool gives the user just the WSDL parts with simple descriptions needed to make his decision without any ambiguity or needless complexity.

CHAPTER FIVE

TOOL ENVIRONMENT

In this thesis, we have investigated the problem of the Web Services understanding, and how to distinguish between the input/output parameters datatypes for the Web Service operations. The result of our research is a tool and its algorithm for extending the XML-Schema to represent the Web Services operations parameters datatypes to reach better comprehension for the Web Service functionality. Unlike previous approaches, which give a semantic for Web Services during its WSDL documents, ignoring uncertainty of operations parameters datatypes declarations, our proposed approach can analyze all the operations of the Web Service and then classify the input/output parameters that operations need.

5.1 Enrichment Algorithm

The tool runs automatically when a requester want to bind with a Web Service doing its process. Finally a new enrichment WSDL document attached, and the requester can examine it. This tool executes its functionality during the proposed enrichment algorithm, shown in Figure (5-1).

```
enr_type_algo
    {
    Input_xsd
        If type = (int, short, long, float, double,
        Boolean, decimal, string, timezone) then
        enr_schema(name, type, type)

        elseif type = (ns*:type, datetime, anytype) then
        enr_schema OperationName(Prname, type,
        provided_type)

        elseif enr_schema OperationName(ParName, type,
        enr_type_func)
    }
```

** ns : Custom Datatype*

Figure (5-1): The proposed enrichment algorithm

This algorithm has three *if statements*. The first one which is the best case of the proposed algorithm gives its output enrichment WSDL document without any *function* call or

communications needs ,while the second *if statement* needs to call a function *(provided_type_func)*. This function returns back to the Web service provider to get the parameter datatype needed for the specific operation and this could be a small drawback of the proposed algorithm since it needs some communications with the providers and it may cause to increase the runtime of the proposed algorithm. This function structure is shown in Figure (5-2).

```
function provided_type_func(string)

 {
   if type = undefined* then
   messagbox contains
   {
     "Please select datatype the parameter datatype"

      Listofdatatypes
         {RadioButton}
         {SubmitButton}

   }
```

** unclear datatype*

Figure (5-2): Provided Datatype Function

The last function which appear in Figure (5.1) is *(enr_type_func)*. By this function the proposed tool can determine which datatype the provider selected for the specific parameter, as shown in Figure (5-3).

```
function enr_type_func(string)

 {
  enr_type, type  string;

  If type = "unsignedbyte" then enr_type = "byte";
  If type = "byte" then enr_type = "sbyte";
  If type = "unsignedint" then enr_type = "uint";
  If type = "unsignedshort" then enr_type = "ushort";
  If type = "unsignedlong" then enr_type =  "ulong";
}
```

Figure (5-3): Enrichment Datatype Function

As result for the propped algorithm we get a new WSDL schema called (Enrichment Schema), shown in Figure (5-4).

```
enr_schema OperationName(ParName, type, enr_type)
{
   <operation OperationName = "n1">
        <input ParName= "n2">
          <type>  type  </type>
          <enr_type> enr_type </enr_type>

          <min_enr_appearance> min </min_enr_appearance>
          <max_enr_appearance> max </max_enr_appearance>

        </input>

        <output OperationNameResponse = "n3">
          <type> type </type>
          <enr_type> enr_type </enr_type>

        </output>
   </operation>
}
```

Figure (5-4): Enrichment Schema

5.2 Visual Implementation for The Proposed Tool *(WSDL_ET)*

In this section we present our proposed tool as visual interfaces screens. We operated the tool as a case implementation for the three datatypes classifications; the output for our tool is a new enrichment XML_Schema with more simplification.

The parameters datatypes is classified in three groups:

1. **Clear Datatypes** : The parameters datatypes are clear and need no modifications. The datatypes appear in the proposed enrichment schema as in the original WSDL document but with more simplification. Figure (5-5) shows the original WSDL document for the *(ClearWebService)* Web Service and shows how the parameters datatypes are implemented inside WSDL.

```
1. <s:element name="ClearWebService">
2.   <s:complexType>
3.     <s:sequence>
4.       <s:element minOccurs="0" maxOccurs="1" name="stringParam" type="s:string"/>
5.     </s:sequence>
6.   </s:complexType>
7. </s:element>
8. <s:element name="ClearWebServiceResponse">.
9. <s:complexType>
10.      <s:sequence>
11.
   <s:element minOccurs="0" maxOccurs="1" name="ClearWebServiceResult" type="s:string"/>
12.       </s:sequence>
13.      </s:complexType>
14.      </s:element>
```

Figure (5-5): WSDL document for (*ClearWebService)* Web service

In line 4 the element *StringParam* has a clear *string* type represented in a formal way, so, no modifications are needed. The enrichment schema for this *StringParam* is show in Figure (5-6) but with more simplifications.

```
enr_schema_ClearWebService ("stringParam", string, string)

{
   <operation OperationName="ClearWebService">
     <input ParName = "stringParam">
      <type>string</type>
      <enr_type>string</enr_type>
      <min_enr_appearance> 0 </min_enr_appearance>
     <max_enr_appearance> 1</max_enr_appearance>
  </input>

  <output OperationNameResponse= "ClearWebServiceResponse">
     <type>string</type>
     <enr_type>string</enr_type>
  </output>
 </operation>

}
```

Figure (5-6): Enrichment schema for (*ClearWebService)* Web service

The enrichment schema in Figure (5-6) consists of three parts, the operation name and its parameters names and also each parameters datatypes with its constraints. These parts help the requester to know what does the Web Service serve, and also what are the parameters datatypes needed.

2. **Indistinguishable Datatypes** : In this case of datatypes, the parameters datatypes is implemented using a custom datatypes *(ns : where ns is .NET namespace)* which is not a formal representation for the datatypes. Figure (5-7) shows a WSDL document for (*IndistinguishableWebService)* Web Service and also shows how the parameters datatypes are implemented inside WSDL.

```
1. <s:element name="IndistinguishableWebService">
2.  <s:complexType>
3.   <s:sequence>
4.     <s:element minOccurs="0" maxOccurs="1" name="byteParam" type="s:unsignedByte"/>
5.   </s:sequence>
6.  </s:complexType>
7. </s:element>

8. <s:element name="IndistinguishableWebServiceResponse">
9.   <s:complexType>
10.      <s:sequence>

11.      <s:element minOccurs="0" maxOccurs="1" name="Indistinguis
   hableWebServiceResult" type="s: unsignedByte "/>
12.      </s:sequence>
13.      </s:complexType>
14.      </s:element>
```

Figure (5-7): WSDL document for (*IndistinguishableWebService)* Web Service

In line 4 the element *ByteParam* have a *unsignedbyte* type which is represented using .NET namespace and that representation is not a formal way. Additionally the Web Service provider has not selected these types, so that these parameters datatypes are needed to represent in a formal and simple representation. The proposed tool is run and converts all these .NET namespace datatypes and presents them in enrichment schema *(Figure 5.3 : Enrichment Datatype Function)*.

The enrichment schema for this *ByteParam* is shown in Figure (5-8) with more specification and more simplifications.

```
enr_schema_IndistinguishableWebService ("byteParam", unsignedByte, Byte)

{
 <operation> OperationName="IndistinguishableWebService">
   <input> ParName = "byteParam">

     <type>unsignedByte</type>
     <enr_type>Byte</enr_type>
     <min_enr_appearance>1</min_enr_appearance>
      <max_enr_appearance> 1 </max_enr_appearance>

   </input>

   <output OperationNameResponse= "IndistinguishableWebServiceResponse">
      <type>unsignedByte</type>
        <enr_type>Byte</enr_type>
   </output>
 </operation>
}
```

Figure (5-8): Enrichment schema for (*IndistinguishableWebService)* Web Service

3. **Unclear Datatypes** : This is the last classification of parameters datatypes. In this case the tool sends an interface application to Web Service provider to determine the selected datatypes, because the tool can not guess which datatypes the provider selected for the specific parameters. Figure (5-9) shows a WSDL document for *(UnclearWebService)* Web Service and also shows how the parameters datatypes are implemented inside WSDL.

```
<s:element name="UnclearWebService">
 <s:complexType>
  <s:sequence>
   <s:element minOccurs="0" maxOccurs="1" name="IntList" type="tns:ArrayOfInt"/>
  </s:sequence>
</s:complexType>
 </s:element>
  <s:complexType name="ArrayOfInt">
   <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="int" type="s:int"/>
   </s:sequence>
  </s:complexType>

<s:element name=" UnclearWebServiceResponse">
  <s:complexType>
    <s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="UnclearWebServiceResult" type="
tns:ArrayOfInt "/>
   </s:sequence>
 </s:complexType>
</s:element>
```

Figure (5-9): WSDL document for *(UnclearWebService)* Web Service

In line 4 the element *IntList* has a *ArrayOfInt* type which is represented using .NET namespace and that representation is not a formal way and is ambiguous, These parameters datatypes need to be presented in a formal and clear representation. The proposed tool is run and converts all these .NET namespace datatypes by returning back to the Web Service provider *(Figure (5-2) : Provided Datatype Function)* and asking to select from a list the datatypes for specific parameters. Figure (5-10) shows the interface which the tool uses to ask the Web Service provider to select the specific datatype.

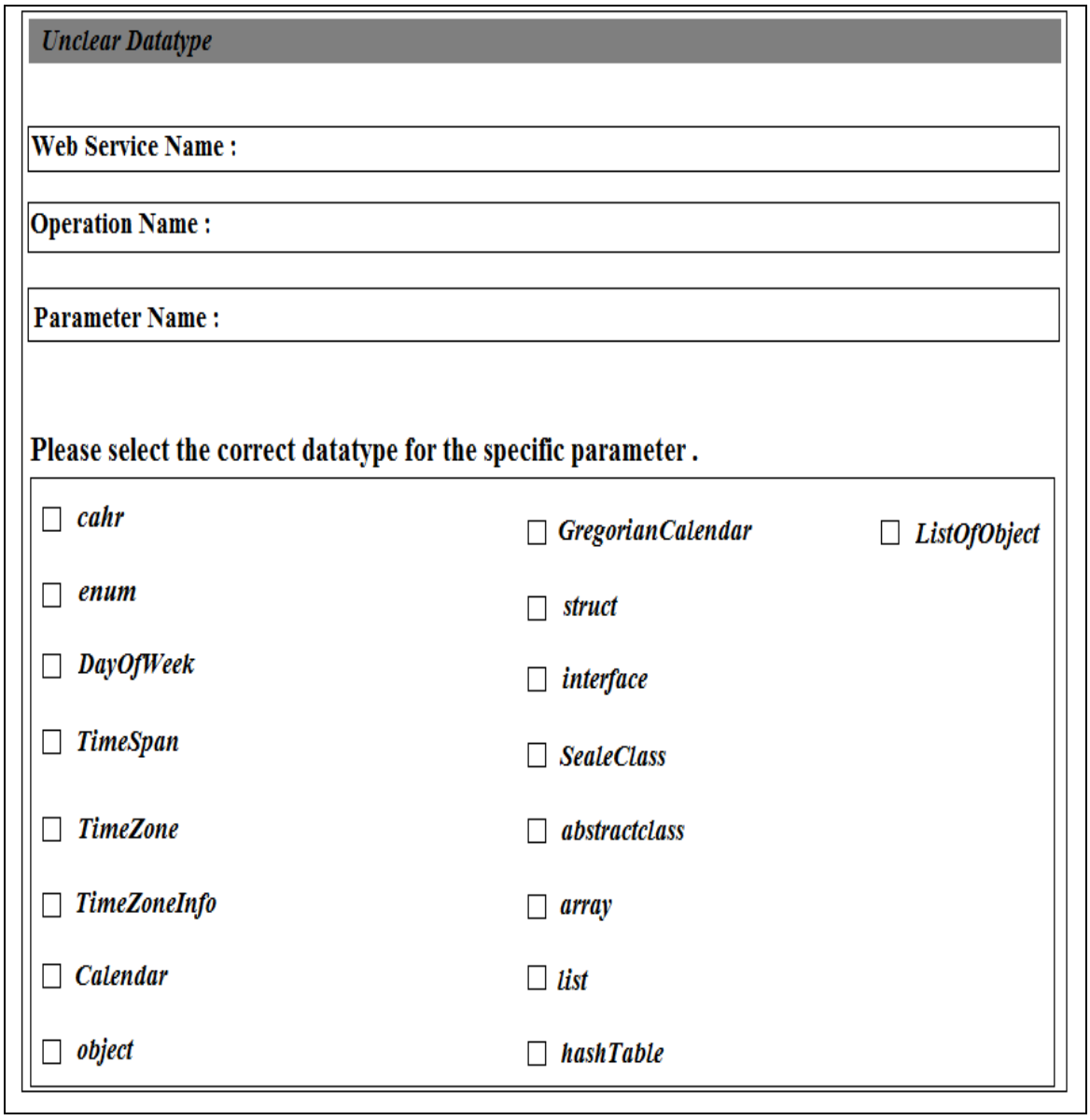


Figure (5-10): Provided Datatype Interface

When the provider selects the specific datatype, the tool starts to create a new enrichment WSDL document with clearer parameters datatypes. The enrichment schema for the WSDL document which appeared in Figure (5-9) is shown in Figure (5-11) after the tool completed its functionality.

```
enr_schema_UnclearWebService ("IntList", ArrayOfInt, List)

{

  <operation OperationName="IndistinguishableWebService">
    <input ParName = "IntList">

      <type>ArrayOfInt</type>
      <enr_type>List</enr_type>

      <min_enr_appearance>0</min_enr_appearance>
      <max_enr_appearance> 1</max_enr_appearance>
    </input>

   <output OperationNameResponse= "UnclearWebServiceResponse">
    <type> ArrayOfInt </type>
    <enr_type> List </enr_type>
   </output>
 </operation>

}
```

Figure (5-11): Enrichment schema for *(UnclearWebService)* Web Service

5.3 Summary

In this chapter we have introduced a full summary of our proposed tool, the tool algorithm, the tool functions, and we have also show that how the tool deals with the three classified datatypes in a visual implementation.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

The output of our thesis is a tool and its algorithm for extending the XML-Schema to represent the operations parameters to reach better comprehension for the Web Service functionality. Unlike previous approaches, which give a semantic for the Web Services during its WSDL documents, ignoring necessary of the operations parameters datatypes declarations, our approach can analyze all the operations of the Web Service and then classify all input output parameters which operations need. The tool deals with 33 datatypes and breaks them into three categories of datatypes discussed in chapter 4.

6.1 Conclusions

One of the problem that still facing Web Service is that the datatypes specification is difficult to understood and reuse by service requester. This thesis had proposed an approach to solve this problem, the approach is based on the following:

1. Analyzing the XSD based datatypes inside WSDL produced by the .NET platform for different prototype Web Service .

2. Classify the datatypes specification into the following Categories:

a. Clear Datatypes : This Category includes the datatypes can easily be understood by service requester.

Table (6-1) shows these datatypes and how they implemented inside WSDL.

Table (6-1): Clear datatypes

	Datatype	Method	Implemented inside WSDL
1	**int**	`public int intExample(Int32 intpar)`	<xs:element minOccurs="0" name="intpar" type="**xs:int**" />
2	**short**	`public short shortExample(Int16 shortpar)`	<xs:element minOccurs="0" name="shortpar" type="**xs:short**"/>
3	**long**	`puplic long longExample (Int64 longPar)`	<xs:element minOccurs="0" name="longPar" type="**xs:long**"/>
4	**double**	`public double doubleExample(Double doublePar)`	<xs:element minOccurs="0" name="doublePar" type="**xs:double**"/>
5	**boolean**	`public bool boolExample(Boolean boolPar)`	<xs:element minOccurs="0" name="boolPar" type="**xs:boolean**"/>
6	**decimal**	`public decimal decimalExample(Decimal decimalPar)`	<xs:element minOccurs="0" name="decimalPar" type="**xs:decimal**"/>
7	**string**	`public string stringExample (String stringPar)`	<xs:element minOccurs="0" name="stringPar" nillable="true" type="**xs:string**" />
8	**timeZone**	`public TimeZone timeZoneExample (TimeZone timeZonePar)`	<xs:element minOccurs="0" name="arg0" type="**tns:timeZone**"/>
9	**float**	`public float floatExample(Single floatPar)`	<xs:element minOccurs="0" name="floatPar" type="**xs:float**"/>

b. Indistinguishable Datatypes : This category include the datatypes that can be distinguished by the proposed tool, Table (6-2) shows these datatypes and how they implemented inside WSDL.

Table (6-2): Indistinguishable datatypes

	Datatype	Method	Implemented inside WSDL
1	**byte**	**byte byteExample(Byte bytepar)**	<xs:element minOccurs="0" " name="bytepar" type="**xs:unsignedByte**" />
2	**sbyte**	**sbyte byteExample(SByte bytepar)**	<xs:element minOccurs="0" name="sbytepar" type="**xs:byte**"/>
3	**uint**	**public uint uintExample(UInt32 uintpar)**	<xs:element minOccurs="0" name="uintPar" type="**xs:unsignedInt**"/>
4	**ushort**	**public ushort shortExample(UInt16 ushortPar)**	<xs:element minOccurs="0" name="ushortPar" type="**xs:unsignedShort**" />
5	**ulong**	**puplic ulong ulongExample (UInt64 UlongPar)**	<xs:element minOccurs="0" name="ulongPar" type="**xs:unsignedLong**"/ >
6	**Class**	<xs:sequence> <xs:element minOccurs="0" name="Name1" nillable="true"type="**xs:string**"/> <xs:element minOccurs="0" name="NickName" nillable="true" type="**xs:string**"/> </xs:sequence>	**13**

c. Unclear Datatypes : This category include the datatypes that is difficult to be understood by the requester and also cannot be distinguished by the proposed approach . Table (6-3) shows these datatypes and how they implemented inside WSDL.

Table (6-3): Unclear Datatypes

	Datatype	Implemented inside WSDL
1	char	<xs:element xmlns:q1="http://schemas.microsoft.com/2003/10/Serialization/" minOccurs="0" name="charInput" type="**q1:char**"/>
2	object	<xs:element minOccurs="0" name="objectPar" nillable="true" type="**xs:anyType**"/>
3	enum	<xs:element xmlns:q5="http://schemas.datacontract.org/2004/07/TeamProject1" minOccurs="0" name="enumPar"type="**q5:DayofWeek**"/>
4	DateTime	<xs:element minOccurs="0" name="datePar" type="**xs:dateTime**"/>
5	DayOfWeek	<xs:element xmlns:q7="http://schemas.datacontract.org/2004/07/System" minOccurs="0" name="dayPar"type="**q7:DayOfWeek**"/>
6	TimeSpan	<xs:element xmlns:q9="http://schemas.microsoft.com/2003/10/Serialization/" minOccurs="0" name="timeSpanPar"type="**q9:duration**"/>
7	Calendar	<xs:element name="arg0" type="**xs:dateTime**" minOccurs="0"/>
8	TimeZone	<xs:element xmlns:q11="http://schemas.datacontract.org/2004/07/System" minOccurs="0" name="timeZonePar"nillable="true" type="**q11:TimeZone**"/>
9	GregorianCalendar	<xs:element name="arg0" type="**xs:dateTime**" minOccurs="0"/>
10	TimeZoneInfo	<xs:element xmlns:q13="http://schemas.datacontract.org/2004/07/System" minOccurs="0" name="timeZoneInforPar"nillable="true" type="**q13:TimeZoneInfo**"/>
11	struct	<xs:element xmlns:q16="http://schemas.datacontract.org/2004/07/TeamProject1" minOccurs="0" name="structPar"type="**q16:ValType1**"/>
12	interface	<xs:element minOccurs="0" name="interfcaePar" nillable="true"type="**xs:anyType**"/>
13	Sealed Class	<xs:element xmlns:q1="http://schemas.datacontract.org/2004/07/TeamProject1" minOccurs="0" name="sealedPar"nillable="true" type="**q1:SealedClass**"/>
14	Abstract class	<xs:element xmlns:q2="http://schemas.datacontract.org/2004/07/TeamProject1" minOccurs="0" name="abstractPar"nillable="true" type="**q2:AbstractClass**"/>
15	Arrays	<xs:element xmlns:q3="http://schemas.microsoft.com/2003/10/Serialization/Arrays" minOccurs="0" name="arrayPar" nillable="true" type="**q3:ArrayOfint**"/>
16	List	<xs:element xmlns:q5="http://schemas.microsoft.com/2003/10/Serialization/Arrays" minOccurs="0" name="listPar" nillable="true" type="**q5:ArrayOfstring**"/>
17	hashTable	<xs:element xmlns:q1="http://schemas.microsoft.com/2003/10/Serialization/Arrays"minOccurs="0" name="hashPar" nillable="true"type="**q1:ArrayOfKeyValueOfanyTypeanyType**"/>
18	List of Object	<xs:element xmlns:q23="http://schemas.datacontract.org/2004/07/TeamProject1"minOccurs="0" name="getTeamsResult" nillable="true" type="**q23:ArrayOfTeam**"/>

3. Enriching the datatypes specification by producing an XML document of the enrichment specification depending on the following :

a. For the clear datatypes it will be remain the same in the result XML document.

b. In the case be indistinguishable datatypes the approach will use rules or conditions to decide the enrichment datatype as explained in the examples of chapter 5 .

c. For the unclear datatypes, in this case the provider is asked to select the prepare the enrichment to be put in the resulted XML. So the provider in intervention is limited to this category of unclear specification.

Based on this thesis approach, a proof of concept tool had been built that can use any WSDL document as input and then produced the enriched datatype specification based on it.

6.2 Future work

Future work will concentrate on the following:

1. This research had depended merely on C# datatypes, however the future work will consider other languages datatypes such as Java or VB., etc.
2. Comparing the datatypes specification inside WSDL documents when using different programming languages to produc Web Services.
3. Enhancing the tool to enable it to work with datatypes specification produced by different programming languages .

In this thesis, we have investigated the problem of the Web Services understanding, and how to distinguish between the input/output parameters datatypes for the Web Service operations.

REFERENCES

Alexandre Bellini, Antonio Francisco do Prado, Luciana Aparecida Martinez Zaina. Top-Down Approach for Web Services Development, Fifth International Conference on Internet and Web Applications and Services, 2010.

Ali Arsanjani, Brent Hailpern, Joanne Martin, Peri L. Tarr, IBM Research Report. Web Services: Promises and Compromises, June 20, 2002.

Chien Hsiang Lee, San Yih Hwang. A Model for Web Services data in Support of Web Service Composition and Optimization, IEEE, 2009.

Eladio Domínguez, Jorge Lloret, Beatriz Pérez, Áurea Rodríguez, Ángel L. Rubio and María A. Zapata. A Survey of UML Models to XML Schemas Transformations, Lecture Notes in Computer Science Volume 4831, pp 184-195, 2007.

Evren Sirin, Bijan Parsia and James Hendler. Composition-driven Filtering and Selection of Semantic Web Services, American Association for Artificial Intelligence, 2004.

Glenford Myers. The Art of Software Testing, ISBN 0-471-04328-1, John Wiley. Neumann, P. (2004). Principled assuredly trustworthy composable architecture.
Hongbing Wang, Joshua Zhexue Huang, Yuzhong Qu, Junyuan Xie. Web services: Problems and Future Directions, Elsevier, 2004.

Houda EL Bouhissi, Mimoun Malki. Reverse Engineering Existing web Services Applications, 16 th Working conference on Reverse Engineering, 2009.

Jan Hausmann Hendrik, Reiko Heckel and Marc Lohmann. Model-based development of Web service descriptions enabling a precise matching concept, International Journal of Web Services Research, Vol.2, No.2, 2005.

Jia Zhang, Liang-Jie Zhang. Editorial Preface: Web Services Quality Testing,
International Journal of Web Services Research, April-June 2005.
Jicheng Fu, Wei Hao, Farokh B. Bastani, and I-Ling Yen. Model-Driven Development: Where Does the Code Come From? , Insights Learned From a Case Study, Fifth IEEE International Conference on Semantic Computing, 2011.

Jinghai Rao, Su Xiaomeng. A Survey of Automated Web Service Composition Methods, In Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004.

John Timm T. E., Gerald C. Gannod. Specifying Semantic Web Service Compositions using UML and OCL, IEEE International Conference on Web Services (ICWS), 2007.

Juan Vara, Valeria De Castro and Esperanza Marcos. WSDL Automatic Generation from UML Models in a MDA Framework, International Journal of Web Services Practices, Vol.1, No.1-2 (2005), pp. 1-12.

Juanjuan Jiang and Tarja Systä. UML-Based Modeling and Validity Checking of Web Service Descriptions, Proceedings of the IEEE International Conference on Web Services (ICWS'05), 2005.

Kuldeep Kumar and Sandeep Kumar. Some Observations on Semantic Web Service Processes, Tools and Applications, International Journal of Computer Theory and Engineering, Vol. 1, No. 1, April 2009.

Mohamad Ibrahim Ladan. Web Services: Security Challenges, IEEE, 2011.

Raluca Bunduchi, Ian Graham, Robin Williams, Neil Pollock and Alison Smart. XML standards and standard settings. The case of XML standards in the NHS Scotland, 9th EURAS conference, Paris, May 12-13, 2004.

Roberto De Virgilio. Meta-Modeling of Semantic Web Services, IEEE International Conference on Services Computing, DOI 10.1109/SCC.2010.22, 2010.

Roy Grønmo, David Skogan, Ida Solheim, Jon Oldevik. Model-driven Web Service Development, International Journal of Web Services Research, 1(4), Oct-Dec 2004.

Samer Hanna and Ali Alawneh. An Approach of Web Service Quality Attributes Specification, Communications of the IBIMA Journal (ISSN: 1943-7765), 2010.

Samer Hanna and Ammer Abu Ali. The Effect of the Platform on Web Services Robustness Testing. Communications of the Applied Computer Science Journal, 11 (2), pp.: 360- 366, 2011.

Samer Hanna. Web services robustness testing, Ph.D, Durham theses, Durham University, (2008).

Suad Alagić, Philip A. Bernstein, Ruchi Jairath. Object-Oriented Constraints for XML Schema, Lecture Notes in Computer Science, 2010.

Suresh Kumar, P.Varalakshmi. Dynamic Web Service Composition based on Network Modeling with Statistical Analysis and Backtracking, International Journal on Web ervice Computing (IJWSC), Vol.3, No.2, June 2012.

W3C ."XML schema part 2 : Dtatypes " . Second edition, W3C Recommendation 9 December 2008.

Weider D. Yu, Aravind, D. & Supthaweesuk, P. Software Vulnarability Analysis for Web Services Software Systems. Proceedings of the 11th IEEE, 2006.

Weijun Sun, Shixian Li Defen Zhang, YuQing Yan. A Model-driven Reverse Engineering Approach for Semantic Web Services Composition, World Congress on Software Engineering, DOI 10.1109/WCSE.2009.403, 2009.

Wei-Tek Tsai, Yinong Chen, Ray Paul. Specification-Based Verification and Validation of Web Services and Service-Oriented Operating Systems , 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 05), Sedona, pp. 139 – 147, February 2005.

الملخص

تعتبر خدمات الويب (Web Services) مهمة جداً و ضرورية لتكامل البرامج و التطبيقات الموزعة و غير المتجانسة . إحدى أهم المشاكل التي تعيق تطور خدمات الويب هي الصعوبة التي يواجهها مزود الخدمة في تمثيل أنواع البيانات الخاصة بمعلمات العمليات من خلال لغة وصف خدمات الويب (WSDL).

هذه المشكلة تسبب ضعف و صعوبة في عملية فهم وظائف خدمات الويب من قبل مستخدم الخدمة، وصعوبة القيام بالهندسة الراجعة العكسية (Reverse Engineering) ، و تحديد فيما إذا كانت خدمة الويب مناسبة الاحتياجات و متطلبات المستخدم أم لا .

في هذه الرسالة سنعمد إلى دراسة هذه المشكلة بدقة ، و تقديم أسلوب جديد للتعاطي مع المشكلة و حلها من خلال توسيع نطاق وصف أنواع البيانات داخل لغة وصف خدمات الويب (WSDL).

هذه الطريقة تعتمد على إضافة وصف أكثر لأنواع البيانات المستخدمة في العمليات المتضمنة في خدمة الويب و كذلك تبسيط لغة وصف خدمات الويب من خلال عرض وصف خدمات ويب غني و سهل الفهم و مبسط (Enrichment Schema) .

المساهمات العلمية الرئيسية لهذه الأطروحة تتلخص بما يلي:

1. دراسة شاملة و معمقة لـ 33 نوع بيانات في لغة C# و كيفية تمثيلها من خلال لغة وصف خدمات الويب (WSDL).
2. تصنيف أنواع البيانات سابقة الذكر إلى ثلاث مجموعات (واضحة ، يمكن تمييزها ، غامضة).
3. تحسين تمثيل 18% من أنواع البيانات المستخدمة في لغة C# و غير المدعومة في لغة وصف خدمات الويب (WSDL) وبشكل مباشر من خلال تقديم وصف غني و مبسط و سهل لخدمة الويب (Enrichment Schema).
4. تحسين مستوى فهم خدمات الويب من خلال الوصف المبسط و الغني لخدمة الويب.

طريقة لتحسين تمثيل أنواع البيانات داخل لغة وصف خدمات الويب (WSDL) من أجل تحسين فهم خدمات الويب

بوساطة
فؤاد سامح علي الشريدة

بإشـراف
د. سامر حنا

قدمت هذه الرسالة استكمالاً لـمتطلبات الحصول على درجة
الـمـاجـستير في علـم الـحـاسـوب

عـمـادة البحث العلمي والدراسات العليا
جامعة فيلادلفيا

أيار 2013