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

Fuad Alshraiedeh
Computer Science Department
Philadelphia University
Amman- Jordan

Samer Hanna
Software Engineering Department
Philadelphia University
Amman- Jordan

Raed Alazaidah
Computer Science Department
Philadelphia University
Amman- Jordan

***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 paper 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 paper 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 schema.**

***Keywords—Datatypes; Understandability; Web Service***

## I. Introduction: 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 [1] defined the Web Service as software components that allow access to functionality via a Web interface network. Additionally, [2] and [3] 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 paper, 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) [1][4]. 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

However, Web Service faces numerous challenges and problems, including, but not limited to the following [5]:

*a) 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. [6] 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. [7] stated that the current methods and technologies cannot ensure Web Service trustworthiness and that for Web Services to grow, researchers must address this challenge.*

*b)* ***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[8] 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. [9] 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.*

In this paper, 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.

Section two discuses related work of web service understandability. In section three we propose the model while in section four brief discussion about the proposed tool. The last section introduces the conclusion and future work.

## II. Related Work: 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 [10][11]. Other researchers recommend a model-driven process for web services development [8], and there are many others.

### A. 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 [12], 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 [11] 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 [13], which in turn are used to perform selections, descriptions, discovery, composition, and interaction with the Web Services [14]. 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 paper is about representing the information inside WSDL in a more understandable form.

In this paper, 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 paper.

### B. Reverse Engineering Approach for Semantic Web Services Composition

Reference [12] 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.

### C. Model-Driven Web Service Development

In this field [15] 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.

### D. Reverse Engineering Existing Web Service Applications

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 [16].

As we illustrated before, a WSDL file is an independent XML-based standard which is proposed by W3C to represent the Web Services functionality [16]. 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), [16] 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) Reference [16] 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 meta model.*

This MIDAS-CASE web service architecture is one of the most important approaches[17] 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.

## III. The Proposed Model: Extending the XML-Schema Datatypes Specification to Reach Better Comprehension of the WSDL

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.

### *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 :

*a) The graphical implementation gives a full summary for the Web Service functionality but with few details.*

*b) The graphical implementation is easier to understand than textual implementation (WSDL document).*

*c) 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 paper. 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 paper, 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:

*a) Do all of the parameters datatypes need to be extended ?*

*b) Can the tool distinguish between the parameters datatypes ?*

*c) How can we reach better comprehension for the Web Services ?*

Section 3.2 shows the model which this paper 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:

*a) Clear Datatypes .*

*b) Indistinguishable Datatypes .*

*c) Unclear Datatypes .*

Figure 1 shows how the tool deals with these datatypes.

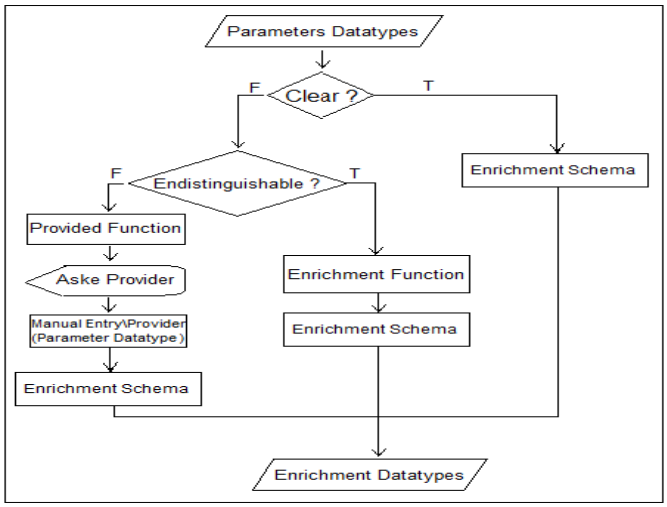


Fig. 1. The datatypes processing

### *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 2 shows the general structure of the proposed model.

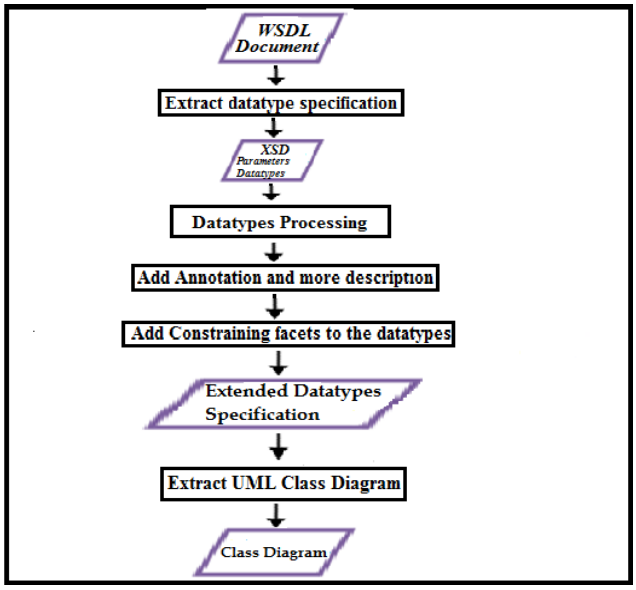


Fig. 2. The proposed model

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.[15]. 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 I. *BYTE* DATA TYPE AND ITS ALIAS (BYTE)

| Short Name | C# Class | Type |
|---|---|---|
| Byte | Byte | Unsigned integer |

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

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

*b) byte in Java :*

TABLE II. *BYTE* DATA TYPE AND ITS WRAPPER (*BYTE*)

| Name | Wrapper Class | Type |
|---|---|---|
| **Byte** | Byte | Signed integer |

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:

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:

a. <element name=*"bytePar"*type=*"xsd:byte"*/>
b. <element name=*"byteExampleReturn"* type=*"xsd:byte"*/>

***2-*** *char* datatype

*a) char in .NET*

TABLE III. *CHAR* DATA TYPE AND ITS ALIAS

| ShortName | .NET Class | Type |
|---|---|---|
| **char** | Char | A single Unicode character |

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:

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" type="**q2:char**"/>

*b) char in Java*

TABLE IV. *CHAR* DATA TYPE AND ITS ALIAS (BYTE)

| Name | Wrapper Class | Type |
|---|---|---|
| **char** | Character | A single Unicode character |

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 insideWSDLas:

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

***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="charPar" type="**xsd:anyType**"/>

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 paper we want to implement our proposed approach on .NET tool as case study.

*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 3 includes three examples of occurrences of a specific element.

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

Fig. 3. min/max occurrence cases

In example 1 Figure 3 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) 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.

*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 we show three groups for parameters datatypes; these types are included in three possible cases.

*A. 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 V. *DOUBLE* DATA TYPE

| Short Name | .NET Class | Type |
|---|---|---|
| **Double** | Double | Double-precision floating point type |

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)

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:

Step 1: Extract the WSDL document for the (public ***double*** doubleExample(***Double*** doublePar)) method ,

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.

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.

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>**

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.

*1) 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

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:

a. <xs:element minOccurs="0" name="Name1" nillable="true" type="**xs:string**"/>
b. <xs:element minOccurs="0" name="NickName" nillable="true" type="**xs:string**"/>

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 .

Example

Public uint uintExample(UInt32 uintpar)

The unit datatype is implemented inside WSDL as:

a. <xs:element minOccurs="0" name="uintPar" type="**xs:unsignedInt**"/>
b. <xs:element minOccurs="0" name="uintExampleResult" type="xs:unsignedInt"/>

The uint datatypes is implemented inside WSDL as *unsignedint* which is a 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.

Step 1: Extract the WSDL document for Web Service.

Step 2:Extract the parameter datatypes XSD as:

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

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>

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>.

Step 5: Class diagram will be generated during any published tool depending on the new enrichment WSDL document.

*2) 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

int[] ArrayExample(int[] arrayPar.

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 ) is defined as:

List<int> ListExample(List<String> listPaList<int>

The WSDL document declaration for the *( list* of *int*) and *(list* of *string*) is shown 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 operation 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:

Step 1: Extract the WSDL document for Web Service.

Step 2: Extract the parameters datatypes XSD as:

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

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.

Step4: Add constraints that provide how many times the input and output parameters will appear as we presented before:

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

Step 5: The new class diagram for enrichment schema will created the simplest and clearest way.

## IV. TOOL ENVIRONMENT

*1) Enrichment Algorithm*

The tool runs automatically when a requester wants 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 4.

```
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
```

Fig. 4. 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.

```
function provided_type_func(string)
 {
  if type = undefined* then
  messagbox contains
  {   "Please select datatype the parameter datatype"
{
       Listofdatatypes
      {RadioButton}
      {SubmitButton}
*  unclear datatype}
```

Fig. 5. Provided Datatype Function

The last function which appears in Figure 6 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 6.

```
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";
}
```

Fig. 6. Enrichment Datatype Function

As result for the propped algorithm we get a new WSDL schema called (Enrichment Schema), shown in Figure 7.

```
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>  }
```

Fig. 7. Enrichment Schema

*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:

*a) 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. The parameters datatypes are implemented inside WSDL as:

a. <s:element minOccurs="0" maxOccurs="1" name=" **stringParam**" type="**s:string**"/>
b. <s:element minOccurs="0" maxOccurs="1" name=" ClearWebServiceResult" type="**s:string**"/>

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 8 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>
  }
```

Fig. 8. Enrichment schema for (*ClearWebService)*

The enrichment schema in Figure 8 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.

*b) 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. WSDL document for (*IndistinguishableWebService)* is implemented inside WSDL as

a. <s:element minOccurs="0" maxOccurs="1" name="**byteParam**" type="**s:unsignedByte**"/>
b. <s:element minOccurs="0" maxOccurs="1" name="IndistinguishableWebServiceResult" type="s: unsignedByte "/>

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. The enrichment schema for this *ByteParam* is shown in Figure 9 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> }
```

Fig. 9. Enrichment schema for (*IndistinguishableWebService)*

*c) 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 cannot guess which datatypes the provider selected for the specific parameters. WSDL document for (IndistinguishableWebService) is implemented inside WSDL as:

a. <s:sequence><s:element minOccurs="0" maxOccurs="1" name="IntList" type="tns:ArrayOfInt"/>
b. <s:element minOccurs="0" maxOccurs="unbounded" name="int" type="s:int"/>
c. <s:element name=" UnclearWebServiceResponse">
<s:element minOccurs="0" maxOccurs="1" name="UnclearWebServiceResult" type=" tns:ArrayOfInt "/>

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 and asking to select from a list the datatypes for specific parameters. Figure 10 shows the interface which the tool uses to ask the Web Service provider to select the specific datatype.

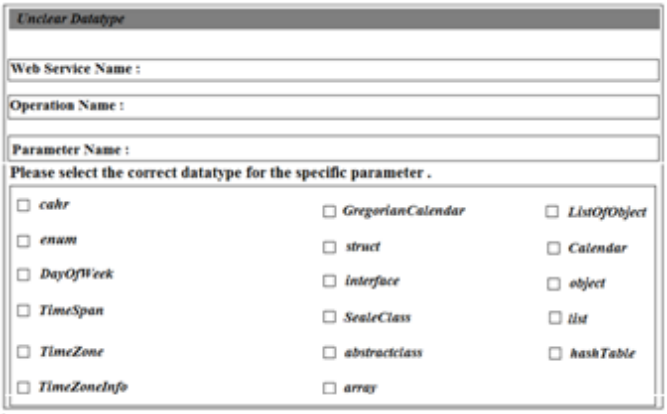


Fig. 10. Provided Datatype Interface

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 8 is shown in Figure 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>
}
```

Fig. 11. Enrichment schema for *(UnclearWebService)*

## V. Conclusion and Future Work

The output of our paper 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.

### A. 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 paper 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 shows these datatypes and how they implemented inside WSDL.

TABLE VI. CLEAR DATATYPES

| Datatype | Implemented inside WSDL |
|---|---|
| Int | <xs:element minOccurs="0" name="intpar" type="xs:int" /> |
| Short | <xs:element minOccurs="0" name="shortpar" type="xs: short"/> |
| Long | <xs:element minOccurs="0" name="longPar" type="xs: long"/> |
| Double | <xs:element minOccurs="0" name="doublePar" type="xs:double"/> |
| Boolean | <xs:element minOccurs="0" name="boolPar" type="xs: boolean"/> |
| Decimal | <xs:element minOccurs="0" name="decimalPar" type="xs:decimal"/> |
| String | <xs:element minOccurs="0" name="stringPar" nillable="true"type="xs:string"/> |
| timeZone | <xs:element minOccurs="0" name="arg0" type="tns: timeZone"/> |
| Float | <xs:element minOccurs="0" name="floatPar" type="xs: float"/> |

*a) Indistinguishable Datatypes : This category include the datatypes that can be distinguished by the proposed tool, Table 7 shows these datatypes and how they implemented inside WSDL.*

TABLE VII. INDISTINGUISHABLE DATATYPES

| Datatype | Implemented inside WSDL |
|---|---|
| Byte | <xs:element minOccurs="0" " name="bytepar" type="**xs:unsignedByte**" /> |
| Sbyte | <xs:element minOccurs="0" name="sbytepar" type="**xs:byte**"/> |
| Uint | <xs:element minOccurs="0" name="uintPar" type="**xs:unsignedInt**"/> |
| Ushort | <xs:element minOccurs="0" name="ushortPar" type="**xs:unsignedShort**"/> |
| Ulong | <xs:element minOccurs="0" name="ulongPar" type="**xs:unsignedLong**"/> |

*b) 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 8 shows these datatypes and how they implemented inside WSDL.*

TABLE VIII. UNCLEAR DATATYPES

| Datatype | Implemented inside WSDL |
|---|---|
| Char | <xs:element xmlns:q1="http://schemas.microsoft.com/2003/10/Serialization/" minOccurs="0" name="charInput" type="**q1:char**"/> |
| Object | <xs:element minOccurs="0" name="objectPar" nillable="true" type="**xs:anyType**"/> |
| Enum | <xs:element xmlns:q5="http://schemas.datacontract.org/2004/07/TeamProject1" minOccurs="0" name="enumPar"type="**q5:DayofWeek**"/> |
| DateTime | <xs:element minOccurs="0" name="datePar" type="**xs:dateTime**"/> |
| DayOfWeek | <xs:element xmlns:q7="http://schemas.datacontract.org/2004/07/System" minOccurs="0" name="dayPar"type="**q7:DayOfWeek**"/> |
| TimeSpan | <xs:element xmlns:q9="http://schemas.microsoft.com/2003/10/Serialization/" minOccurs="0" name="timeSpanPar"type="**q9:duration**"/> |
| Calendar | <xs:element name="arg0" type="**xs:dateTime**" minOccurs="0"/> |
| TimeZone | <xs:element xmlns:q11="http://schemas.datacontract.org/2004/07/System" minOccurs="0" name="timeZonePar"nillable="true" type="**q11:TimeZone**"/> |
| GregorianCalendar | <xs:element name="arg0" type="**xs:dateTime**" minOccurs="0"/> |
| TimeZoneInfo | <xs:element xmlns:q13="http://schemas.datacontract.org/2004/07/System" minOccurs="0" name="timeZoneInforPar"nillable="true" type="**q13:TimeZoneInfo**"/> |

| | |
|---|---|
| Struct | <xs:element xmlns:q16="http://schemas.datacontract.org/2004/07/TeamProject1" minOccurs="0" name="structPar"type="**q16:ValType1**"/> |
| Interface | <xs:element minOccurs="0" name="interfcaePar" nillable="true"type="**xs:anyType**"/> |
| Sealed Class | <xs:element xmlns:q1="http://schemas.datacontract.org/2004/07/TeamProject1" minOccurs="0" name="sealedPar"nillable="true" type="**q1:SealedClass**"/> |
| Abstract Class | <xs:element xmlns:q2="http://schemas.datacontract.org/2004/07/TeamProject1" minOccurs="0" name="abstractPar"nillable="true" type="**q2:AbstractClass**"/> |
| Arrays | <xs:element xmlns:q3="http://schemas.microsoft.com/2003/10/Serialization/Arrays" minOccurs="0" name="arrayPar" nillable="true" type="**q3:ArrayOfint**"/> |
| List | <xs:element xmlns:q5="http://schemas.microsoft.com/2003/10/Serialization/Arrays" minOccurs="0" name="listPar" nillable="true" type="**q5:ArrayOfstring**"/> |
| Hash Table | <xs:element xmlns:q1="http://schemas.microsoft.com/2003/10/Serialization/Arrays"minOccurs="0" name="hashPar" nillable="true"type="**q1:ArrayOfKeyValueOfanyTypeanyType**"/> |
| 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 paper 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.

### B. 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 produce Web Services.*

*3) Enhancing the tool to enable it to work with datatypes specification produced by different programming languages.*

## ACKNOWLEDGMENT

The authors thank all the professors in faculty of information technology in Philadelphia University, especially prof. Said Ghoul and Dr. Nameer Emam for their scientific and accurate help.

## REFERENCES

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

[2] Hongbing Wang, Joshua Zhexue Huang, Yuzhong Qu, Junyuan Xie. Web services: Problems and Future Directions, Elsevier, 2004.

[3] 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.

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

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

[6] 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.

[7] Jia Zhang, Liang-Jie Zhang. Editorial Preface: Web Services Quality Testing, International Journal of Web Services Research, April-June 2005.

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

[9] Glenford Myers. The Art of Software Testing, ISBN 0-471-04328-1, John Wiley. Neumann, P. (2004). Principled assuredly trustworthy composable architecture.

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

[11] 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.

[12] 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.

[13] 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.

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

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

[16] 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.

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