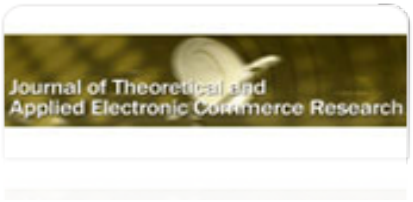

Journal of Theoretical and Applied Electronic Commerce Research
E-ISSN: 0718-1876
ncerpa@utalca.cl
Universidad de Talca
Chile

Hanna, Samer
An Approach to Modeling Web Services Datatype Descriptions
Journal of Theoretical and Applied Electronic Commerce Research, vol. 11, núm. 2, mayo, 2016, pp. 64-82
Universidad de Talca
Curicó, Chile

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 11 / ISSUE 2 / MAY 2016 / 64-82

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

An Approach to Modeling Web Services Datatype Descriptions

Samer Hanna

Philadelphia University, Department or Software Engineering, Jerash, Jordan, shanna@philadelphia.edu.jo

Received 20 February 2015; received in revised form 12 July 2015; accepted 1 September 2015

Abstract

Web services are becoming a significant part of Web applications in different fields such as e-commerce applications. A problem that is facing Web services is that the Web services datatype descriptions inside Web Services Description Language are difficult to be understood by service providers or requesters due to many reasons; one of these reasons is the lack of expressiveness of the Schema based datatype system that is used to describe provider side datatypes inside Web Services Description Language. This problem leads to producing vague, custom and inconsistent datatype descriptions by different Web services development techniques inside the auto-generated Web Services Description Languages. This paper proposes an approach to formally model Web services datatype descriptions in order to solve the previous problem. The approach is based on mapping the datatype section inside a Web Services Description Language into a more understandable tree and models. Following the paper's approach, service requesters can understand the datatype descriptions of the provided operations by a Web service inside Web Services Description Language and consequently this will make it easier to invoke these operations. A prototype tool has been built and it proved to be efficient in enhancing Web services understandability.

Keywords: Web services, Web services modeling, Web services datatype descriptions, Extensible markup language schema datatypes, Web services description language, Understandability

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 11 / ISSUE 2 / MAY 2016 / 64-82

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

1 Introduction

Web services is a new paradigm for building distributed application based on distributed heterogeneous services. The future Internet will be based on services, and this new trend will have a significant impact on domains such as e-commerce [24]. *Web services* plays an important role in the e-commerce domain because it facilitates the interaction between the applications in this domain and heterogeneous services in order to fulfill user needs. There are many examples of Web services being used by an e-commerce application; one of these examples is given in [33] of using Web services to improve the performance of the recommender systems that are used in almost all the major e-commerce sites to help customers to find the products they would like to purchase according to some strategy. In this research, heterogeneous local models on different nodes are transformed into models with Web services form in order to improve the performance of recommendation process. Another example is the Web service based dynamic e-commerce, which is regarded as the next evolutionary step of e-commerce in the industry [35].

E-commerce firms are turning to Web services to streamline operations and solve the interoperability problem. A Web service, for instance, could carry out payment validation by taking details from one system and initiate a request for other information, such as a credit card account number or expiration date.

Web services frees system developers to concentrate on enabling their business and their customers rather than deal with interoperability headaches by writing glue code and patching systems together [11].

Although Web Services' main objective is to solve the interoperability issue of many fields such as e-commerce based applications; Web services still face some problems; one of these problems is that the Web services datatype descriptions inside Web Service Description Language (WSDL) [4] are difficult to be understood by system developers and this will influence Web services understandability quality attribute. Web services understandability in this paper refers to the effort required by developers to determine what operations from a Web service they need to use and how they should invoke it [7]. Lack of understandability by developers will also affect the interoperability quality because developers will find it difficult to integrate a Web service with their application if they cannot understand the description of his service.

Many solutions had been proposed to the Web services understandability problem. One of the solutions is based on modeling the information inside a WSDL interface to convert it to a more understandable form. Examples of such research are the research to map WSDL to Unified Modeling Language (UML) model [12] and also the research to map WSDL to Ontology such as the research reviewed in [30].

None of the model based solutions to the understandability problem in the literature considered modeling Web services datatype descriptions inside WSDL. As such, this paper considers modeling the XML Schema (XS) [21] based Web services datatypes inside WSDLs to convert it to a more understandable model for service requesters and providers. Service providers and requesters in this research are defined in Section 2.1.

Web services datatype descriptions inside WSDLs have many problems such as: a) the XS definitions based datatype system, used by Web services, has very limited number of datatypes where many of the Web services provider side datatypes e.g. Java or C# datatypes, are not supported by XS, this problem will be discussed in Section 4, b) Web services providers depend, most of the time, on tools to generate the WSDL descriptions for their services and these tools can produce vague, custom and inconsistent datatype specification as will be discussed in Section 4 also.

As an example of the limited number of datatypes in XS, the *array* datatype, which is a basic and central datatype that is used in most of the applications, is not included among the XS datatypes. If a provider side operation has an *array* input or output, it is described inside WSDLs either using the *maxOccurs=unbounded* XS constraining facet [21] and this approach is followed by the Java based Web services development techniques, or using a custom array datatype where custom here means a datatype that belongs to a certain programming language, and this approach is followed by the *.NET* based Web services development techniques. Both of these approaches have many disadvantages as will be discussed in details in Section 4.

Since the main reason for e-commerce, e-business, and other Web-based applications to use Web services is to facilitate the interoperability among heterogeneous applications [11]; Web services datatypes must be understood by those applications in order to enable such applications to invoke the required Web services operations properly regardless of the platform used to build these Web services. Accordingly; failing to describe Web services datatypes in an understandable way to other applications will hinder the existence of Web services itself.

The main objective of this research is to solve the problem of Web service datatype descriptions by suggesting an approach to map Web services datatypes to a more understandable form to service requesters and providers. The mapped form will be based on tree model and XML model as will be discussed in Section 5.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

The research that considered mapping the information inside WSDL to a more understandable model such as using UML class diagram has: (a) focused on modeling Web service composition rather than Web services datatypes, and (b) not considered Web services datatypes problems that will be discussed in Section 4. The research in this paper focuses on modeling Web services datatype taking into consideration these datatype descriptions' problems.

The contributions of this paper are:

- Discussing a problem that is currently affecting Web services understandability and interoperability which is vague datatype descriptions.
- Describing the shortcomings of different Web services development techniques in describing Web services provider side datatypes inside WSDL (Section 4).
- Introducing a novel approach to model Web services datatypes (section 5).
- Introducing a proof of concept tool that can be used by Web services providers and requesters to map WSDL documents to a more understandable model (Section 7).

A background about the main keywords used in this work is introduced in Section 2. The related work to this research is introduced in Section 3. Web services datatype description problems such as: XS datatypes lack of expressiveness and producing vague, inconsistent and custom datatype descriptions by Web services development techniques are discussed in Section 4. Modeling Web services datatype descriptions, which is the proposed solution to the problems in Section 4, is discussed in Section 5. The usefulness of the proposed solution is demonstrated through a case study in Section 6 and also by implementing and using a proof of concept tool in Section 7. Finally the research is finished with conclusion and future work in Section 8.

2 Background

The Background section will first define Web services and Web services development process in Section 2.1, after that, the sample Web services development techniques considered in this research will be discussed in Section 2.2, and finally the XS datatypes will be discussed in Section 2.3.

2.1 Web Services

Web services is an XML based framework introduced by World Wide Web Consortium (W3C) that defines a standard for achieving smooth communication among discrete application systems [8]. Web services is based on the exchange of messages or documents in XML over standard internet protocols such as: Simple Object Access Protocol (SOAP), WSDL, and Universal Description, Discovery and Integration (UDDI).

The process followed in developing Web services applications, in this research, is as follows:

- A service provider who is an application developer that uses one of different Web services development techniques (such as those discusses in section 2.2) to: (a) writes the code of a service in a certain programming language such as Java or C#, depending on the platform of the development technique. (b) Use the development technique to auto-generate the WSDL document corresponding to the operations provided by the service.
- A service provider can then, optionally, publish the auto-generated WSDL by the used development technique, into a service registry such as UDDI.
- A service requester, who is another application developer that wants to integrate a Web service built by a provider in his application, searches for a WSDL description with operations that satisfies the requirement of the application being built.
- The information inside WSDL, such as the input and output datatype descriptions of the provided operations, is used by the service requester to invoke the described Web service build by service provider.

It must be noted that service provider and service requester can also be software components; however, in this research, both are assumed to be human.

2.2 Web Services Development Techniques

Web service providers (developers) depend on Web services development techniques/tools/frameworks to implement and publish their Web services as described in Section 2.1. There are many Web services development

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

techniques that are based on different platforms such as: Microsoft's *.NET* platform, *Oracle's Java Enterprise Edition (EE)*, and *Apache Axis2* [13].

Microsoft Visual Studio is used to develop Web services based on the *.NET* platform; the software development kit for developing and deploying services on *.NET* is called *Windows Communication Foundation* (*WCF)* [5]. On the other hand, many tools and IDEs facilitate the development of Web services based on the *Java EE* platform and Axis2 such as *Eclipse Integrated Development Environment (IDE) for Java EE Developers* and *NetBeans*.

2.3 XML Schema Definitions Datatypes

XS definitions (XSD) datatypes are used by Web services to specify the datatype of the input and output parameters of the provided operations.

XSD datatypes can be divided into:

- Simple datatypes that include:
  - Built-in primitive datatypes: examples of built-in primitive simple datatypes: XS:string, XS:float and XS:decimal.
  - Derived from built-in primitive datatypes: these datatypes are derived from the built-in primitive datatypes by applying some default constraints, for example XS:nonPositiveInteger is derived from integer by restricting the value space of integer to only negative numbers.
  - 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 datatypes) using constraining facets such as minInclusive and maxInclusive that specify the minimum and the maximum values allowed in a datatype.
- Complex datatypes that consist of one or more elements and attributes of simple datatypes.

Figure 1 is an example of an XML Schema based complex datatype that is used to describe a user defined *book* class (Site 1).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
       <xs:element name="book">
              <xs:complexType>
                <xs:sequence>
                     <xs:element name="title" type="xs:string"/>
                     <xs:element name="author" type="xs:string"/>

       <xs:element name="character" minOccurs="0" maxOccurs="unbounded">
                     <xs:complexType>
                       <xs:sequence>
                            <xs:element name="name" type="xs:string"/>
                         <xs:element name="friendOf" type="xs:string" minOccurs="0"
                       maxOccurs="unbounded"/>
                            <xs:element name="since" type="xs:date"/>
                            <xs:element name="qualification" type="xs:string"/>
                       </xs:sequence>
                     </xs:complexType>
                     </xs:element>
                </xs:sequence>
              <xs:attribute name="isbn" type="xs:string"/>
           </xs:complexType>
       </xs:element>
</xs:schema>
```

Figure 1**:** Example of an XS Complex datatype

The XS Complex datatype in Figure 1 specifies the *book* datatype by specifying that it includes a sequence of *title, author, and isbn* elements of simple datatype *string*. The *book* complex datatype includes also sub-complex datatype called *character* that has four elements: *name, friendOf, qualifications* of simple datatype *string*, and *since* element of simple datatype *date*. The cardinality (i.e. the number of possible occurrences) of the *character* complex datatype is specified using the *minOccurs* (the minimum number of occurrences) and *maxOccurs* (the maximum number of occurrences) here *maxOccurs* is set to unbounded which means that there can be as many occurrences of the

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

character element as the *author* wishes. Similar complex datatypes can be found for example in a WSDL definition of a Web service that is used in a book shopping e-commerce application.

The main goal of this research is to map the simple and complex XS datatypes inside WSDLs to a more understandable form in order to enhance the understandability and interoperability of Web services and hence reduce the needed effort by e-commerce and other applications developers, to integrate the described Web services with their applications.

3 Related Works

The related works to this research can be categorized into the following categories:

- Modeling and analyzing Web services specifications by mapping WSDL to UML and other models.
- Solving WSDL problems (faults, or bad practices) such as lacking comments inside WSDL and XS wild-card constructs such as XS: anyType.
- Using semantic based technologies for Web services annotations.
- Using Web service in the e-commerce applications domain.

The most related literature in each of these domains will be analyzed below:

Modeling and analyzing Web services specifications by mapping WSDL to UML and other models is done to convert WSDL documents into a more understandable form to service requesters and providers. Among the research considered mapping WSDL to UML and other models:

- Gronmo et al. [12] was one of the first works in the field of modeling Web services with UML, they proposed an approach to model Web services as a conceptual UML models without using WSDL-specific constructs.
- Jiang & Systa [14] proposed a tool that can be used to transform WSDL documents to its UML class diagram representation.
- Sun et al. [27] proposed a Model Driven Architecture (MDA) based approach for transforming WSDL to UML models and then integrating the generated UML models into composite Web service.
- Sheng et al. [26] proposed an approach for modeling Web services that distinguishes operational behavior, which defines the business logic underpinning the Web service's functioning, and control behavior, which guides the operational behavior's execution progress by identifying the actions to take and the constraints to satisfy.
- Melo and Silveira [17] analyzed Web services messages simple and complex datatypes together with the XS constraining facets and then described an approach to generate test cases for Web services based on perturbing the data value of Web services input based on the analyzed datatype.
- Yu et al. [34] proposed an approach to support the systematic development of dynamically adaptive services oriented systems based on Model Driven Development (MDD).
- Fang et al. [10] proposed a model for Web services specifications which enhance WSDL specification based on different rules that enable service provider to verify specifications and help user to plan correctly the interaction with Web services by reasoning according to the exposed rules.
- Rai et al. [22] proposed an approach for modeling and verifying Web services composition based on recursive composition based algebra and canonical sets.

The objective of the research concerned with solving WSDL faults is to remove ambiguity from WSDL documents and convert it to more understandable forms. Among the research considered solving WSDL faults:

- Pasley [19] discussed the negative side effects to Web services understandability when using the XML Schema wild-card constructs-XS: anyType and XS: anyAttribute- to define Web services datatypes. He mentioned that using such constructs increases complexity and results in ambiguous interface definitions.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

- Crasso et al. [7] described the most common errors that were found in real WSDL documents (e.g. ambiguous names) and explained how these errors impact service discoverability and understandability.

They mentioned that service providers should always *revise* service descriptions to enhance discoverability and understandability. It has been shown that publically available WSDL documents do not contain useful piece of information making Web services syntactic registries ineffective.

According to this research experiments, the inappropriate or lacking comments bad practice existed in more than 70 percent of the experimented WSDL dataset. The results of this research show that we can expect that removing detected bad practice will result in improvement to WSDL documents.

Similar to Crasso et al., Rodriguez et al. [23] also discussed Web services descriptions bad practices that hinder Web service discoverability and understandability. Examples of such practices are: ambiguous names, inappropriate or lacking comments, etc. However, this research studied also how to solve the discoverability and understandability caused by such bad practice. The research proved by experimenting 391 WSDL documents, the usefulness of the proposed approach of improving WSDL descriptions in terms of discoverability and understandability.

It was stated that placing comments within a WSDL file makes the intended functionality of its associated service more understandable. Besides, syntactic service registries exploit this kind of documentation to support discovery.

In this research, the bad practice (called anti-pattern also) of *inappropriate or lacking comments,* occurred in 289 of the 391(roughly 74 percent) experimented WSDL documents. It was concluded that removing this bad practice contributed to service discoverability more than removing any other bad practice.

Coscia et al. [6] proposed an approach to avoid and remove WSDL descriptions bad practices discussed in Rodriguez et al. [23] and Crasso et al. [7]. The approach is based on refactoring a Web service in order to remove its WSDL bad practices (anti-patterns). Refactoring is based on a group of object oriented metrics such as: Coupling between objects, weighted methods per class, abstract type count, and empty parameter methods.

Refactoring focus on modifying structural aspects of services codes such as splitting a service into adding/removing parameters and so on.

Blake and Nowlan [3] discussed the inconsistencies in service-based interface descriptions and message names. The authors detected *naming tendencies* in 596 WSDL documents and proved empirically that these tendencies negatively impact Web services irretrievability.

Some research considered mapping WSDL to ontology based models because WSDL documents describe only the signature of the provided function and do not include semantic information about the described service so the research in this category aim at providing such semantics. Among the research considered mapping WSDL to ontology or adding semantics to Web services:

- Tosi and Morasca [30] presented a systematic literature review that summarizes the current state-of-art for supporting the semantic annotations of web services.

- Alférez et al. [1] proposed a semantically rich variability model to support the dynamic adaptation of Web services compositon.

- Sabou and Pan [25] reported an ongoing work and ideas on how to use techniques developed in the context of semantic Web to improve the current situation in the process of automating Web services discovery and composition.

- Paulraj [20] produced process model ontology of Ontology Web Language-Service (OWL-S) to describe the input and output of a service to facilitate Web services discovery.

- EL Bouhissi et al. [9] proposed a reverse engineering approach to specify Web service according to the Web Service Modelling Ontology (WSMO).

- Talantikite et al. [29] proposed a model of semantic annotations for Web service discovery and composition based on inter-connected network of semantic Web services description in OWL-S.

- Virgilio [32] proposed a meta model-based approach to describe the semantics associated to a Web service based on its WSDL. The semantic WSDL-based description is used to allow interoperability at different levels of abstraction.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

- Bellini et al. [2] proposed creating a meta or abstract model for WSDL in order to allow the developer to shape the service in a more friendly way.

- Kumar et al. [15] proposed a framework to enhance the web service matching to discover relevant web service, which is requested by the user. The framework is based on the semantic description of the web services.

- Rodríguez-García et al. [24] proposed a semantically-enhanced platform that assists in the process of discovering cloud services that best match user needs.

There is also some research in the literature that tackles improving the efficiency of e-commerce applications with the help of Web services technology. Example of such research:

- Zhang and Liu [35] proposed a context-aware Web service composition model for dynamic e-commerce applications to improve the efficiency of the dialogues between users and systems.

Yan-xin et al. [33] proposed a framework for e-commerce recommender Web service system which is a typical software solution used in e-commerce for personalized services. They built a model to specify Web services based on: (1) Input and output interfaces describing names and types of model's input and output variables that may be simple or complex datatypes. (2) Web services methods and (3) WSDL dictionary that provide description of Web services methods in details.

Taconet and Kazi-Aoul [28] proposed a meta-model for defining context-aware application models following the Model Driven Engineering (MDE) approach. The solution is illustrated by modeling a context-aware e-commerce application.

The research in this paper is different than the research in the above literature in that:

- It does not only model Web services descriptions, but also considers the faults in these descriptions.

- It considers the inefficiencies of different Web services development techniques in generating Web services datatype descriptions.

4 Web Services Datatype Descriptions Problems

Web services depend on XS datatypes to describe the datatypes of the provided operations. In the Web services development process discussed in Section 2.1; when a service provider (developer) implements the provided operation of a Web service in *Java* or *C#* for example, a rich datatype system with many datatypes can be used with the provided operations input and output operations parameters, for example, if an operation has a collection input and the provider is using a *.NET* based development technique such as *WCF* and *C#* language, provider can use different collection based datatypes such as *Array*, *ArrayList*, *LinkedList*, etc. However, inside the WSDL description of that Web service, the provider side datatype must be serialized or mapped to an XS based datatype and this will initiate many problems because the XS datatypes are limited (Section 2.3) and many of the provider side datatypes has no XS based counterparts. In the same example of using a collection datatype such as *Array* or *ArrayList* by a service provider, XS has no equivalent of any of these datatypes and consequently WSDL documents may use one of two approaches: (a) depend on the cardinality attribute (*maxOccurs*) to describe collection datatypes or (b) use custom datatype (e.g. *ArrayofInt*) that does not belong to XS datatypes. Both of these approaches will affect negatively Web services understandability and interoperability as will be explained below.

Because of the lack of expressiveness problem of XS datatypes, Web services development techniques may face difficulties when serializing provider side datatypes to their equivalent XS based datatypes inside WSDLs. These difficulties lead to producing vague, custom, identical and inconsistent datatype descriptions. These problems will be discussed in Section 4.1 to 4.4 together with examples of each problem.

4.1 Vague Datatype Descriptions

Web services development techniques map or serialize some provider side datatypes to vague datatype descriptions inside WSDL. The word vague here means a description that is difficult to be understood by service requesters. It is important for the service requester to understand the provider side datatype in order to use the correct datatype when invoking the described operation of a Web service.

As an example of the problem when using the *Java EE* 6 platform to build a Web service: the provider side *Java char* datatype is mapped by *Java EE* platform to the XSD datatype *unsignedShort* inside the auto-generated WSDL, this specification is vague because it will be difficult for the service requesters that only use WSDL, to understand that the provider side datatype is *char* depending on such specification.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

An example of the problem when using the *Apache Axis2* engine to build a Web service: *TimeZone* datatype is mapped to the XS datatype *anyType*. The *anyType* is considered vague because it will be difficult for a service requester to figure out what is the provider side datatype based on this description.

An example of the problem when using the *NET 4.5* platform and *WCF* framework to build a Web service: the .NET datatype *Interface* is mapped by WCF to the XSD *anyType* inside WSDL.

All of the previous problems happen due to the lack of expressiveness of the XS datatype problem and they lead to producing vague and incomprehensible datatype descriptions inside WSDLs.

4.2 Custom Datatype Descriptions

Development techniques may map the provider side datatypes that are not supported by XS to a custom datatype depending on the used platform. Custom here means a datatype that belongs to a certain platform or programming language only and not to the XS datatypes.

As an example of the problem when using the *Java EE 6* platform to build a Web service: a provider side *TimeZone* datatype is serialized to a custom *TimeZone* datatype inside WSDL.

An example of the problem when using the *Apache Axis2* engine to build a Web service: a provider side *ArrayList* datatype is serialized to a custom *ArrayList* datatype inside WSDL.

An example of the problem when using the *WCF* framework with *.NET 4.5* platform to build a Web service: *TimeSpan* is serialized to a custom *TimeSpan* datatype.

The datatype descriptions inside WSDL must be platform independent in order to enable the described service to be understood and used by different platform. Using custom datatypes may reduce the interoperability of the described Web service because such types may not be understood by Web services built by different techniques and also this will increase the difficulty of composing such Web services with other Web service that are building using different techniques.

4.3 Identical Datatype Descriptions

Development techniques may map more than one datatype to an identical description inside WSDL. This problem also will make it difficult for service requester to find out which datatype to use when invoking the described Web service.

As an example of this problem when using the *Java EE 6* platform to build a Web service: the *Java List, ArrayList, and LinkedList* are all mapped inside WSDL to an identical description as the one that is generated when mapping an *array*. So, all of the previous datatypes have identical descriptions inside WSDL. When a service requester encounters this identical description, it will be difficult to conclude if the described provider side operation accepts a *List, ArrayList, LinkedList, or an array*.

An example of the problem when using the *Apache Axis2* engine to build a Web service: *List* datatype is mapped to identical description as the one that is generated when mapping an *array*. Also *HashTable* is mapped in an identical way like the *Map* datatype.

An example of the problem when using the WCF framework with .NET 4.5 platform to build a Web service: *List* and *LinkedList* datatypes are mapped in an identical way of mapping *arrays*

When many provider side datatypes serialized by a development technique to an identical description inside WSDL, the difficulties faced by service requesters in concluding the suitable datatype to use when invoking the described Web service will be increased.

4.4 Inconsistent Datatype Descriptions

Web services development techniques in Section 2.2 may generate different datatype descriptions even when describing the same provider side datatype. This indicates that the XS based datatype descriptions inside WSDL are affected by the development technique and platform used to generate these descriptions.

As an example of this problem, a provider side *char* datatype is serialized to a custom *char* datatype, inside WSDL, when using the WCF framework to generate this WSDL description. While if the Java EE 6 platform is used to generate WSDL then the provider side *char* datatype is serialized to the XS datatype *unsignedShort*.

Another example for this problem is when a provider side operation has an *array* input or output where the *WCF* framework will serialize this provider side *array* to a custom datatype called *ArrayOf* concatenated to the datatype of

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 11 / ISSUE 2 / MAY 2016 / 64-82

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

the array elements while the *Java EE* 6 platform will serialize the *array* using the XS *maxInclusive* constraining facet by giving it the value of *unbounded*.

Datatype descriptions inconsistencies contribute to the main problem in this research which is the lack of understandability of WSDL datatype descriptions generated by different Web services development techniques and platforms.

5 Modeling Web Services Datatype Descriptions

To solve the problems of vague, custom, identical and inconsistent datatype descriptions discussed in Section 4, this research proposes a novel approach of modeling the WSDL's XS based datatype descriptions to map it into a more understandable form to service requesters and providers.

The approach is novel because it conceders the problems discussed in Section 4 when modeling Web services datatypes inside WSDLs. The resulted model distinguishes vague, custom, identical and inconsistent datatype descriptions so that these datatypes can be handled by the service provider of the described Web service. For example service provider can attach comments or annotations to such datatype descriptions inside WSDL in order to make it clear for service requesters the datatype that must be used when invoking the described operations.

Section 5.1 describes the approach used by this research to model Web services datatypes while Section 5.2 will demonstrate the approach used to distinguish vague, custom, identical and inconsistent datatype descriptions.

5.1 Tree Based Model of Web Services Datatype Descriptions

Almost all Web services XML data models proposed in the literature represented an XML document using a tree structure [16], for this reason, a tree structure will be followed in Definition 1 below to map a Web service datatype to a more understandable form and to give a basis for defining the needed rules to distinguish the vague, custom, identical and inconsistent datatype descriptions.

Converting a complex schema based specification to a more understandable tree model is borrowed from the approach followed in the Model Driven Development (MDD) [31].

Definition 1: (Web service complex datatype tree) A Web service complex datatype is a Tree *T*, the root of *T*, which is called *Complex Type* (*CT)*, is the complex datatype name attribute value. The elements belonging to the complex datatype represent the children of the root element *CT*, each one of these children (nodes) is called E_i where $i=1..n$ and *n* is the number of elements belonging to the complex datatype. Each child is represented as an attribute-value pair where the attribute is the value of the *name* attribute of the element of the complex datatype and the value is the *value* of its *type* attribute. So if a complex datatype has n elements then the root of *T* will have *n* children, these children will be called $E_1, E_2,...,E_n$. Constraining facets related to each E_i will also be represented as children of the E_i node in the tree, each of these children will be called C_p where $p=0..m$ where *m* is the number of constraining facets related to a given E_i. Each of C_ps will also be represented as an attribute-value pair where the attribute is the constraining facet name [1] and the value is the value of the constraining facet.

Figure 2 below give the details of Definition 1 for the complex datatype tree.

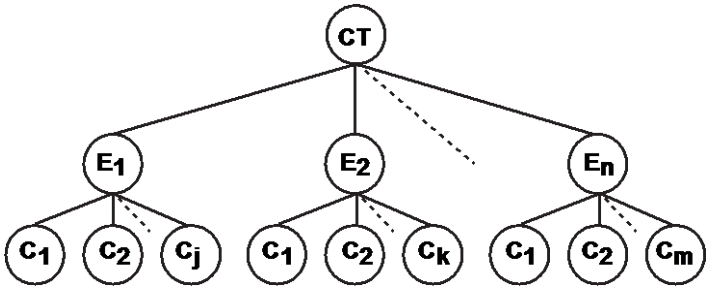

Figure 2: Complex datatype tree

In Figure 2:

CT is a complex datatype with n elements

E_1, E_2,...., E_n are the elements of the complex datatype CT.

C_1, C_2,..., C_j are the constraining facets of the element E_1. C_1, C_2,..., C_k are the constraining facets of the element E_2 and C_1, C_2,..., C_m are the constraining facets of the element E_n.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 11 / ISSUE 2 / MAY 2016 / 64-82

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

Note that there is no relation between j, k and m in Figure 2 since the number of constraining facets for each element is independent of the number of the constraining facets for other elements of the same complex datatype.

E_i.attribute is the value of the name attribute of the i^{th} element of the specified complex datatype CT.

E_i.value is the value of the type attribute of the i^{th} element of the specified complex datatype CT.

$E_i.C_j$.attribute is the name of the j^{th} constraining facet of the i^{th} element of the specified complex datatype CT.

$E_i.C_j$.value is the value of the j^{th} constraining facet of the i^{th} element of the specified complex datatype CT.

If the specified complex datatype *CT* has more than one attribute then an associative array will be attached to the root of the tree where each row in this tree will represent an attribute and its value.

Suppose that a complex datatype *CT* has two attributes; first attribute called name and its value is *Name* and the second attribute called value and its value is *Value* then these specifications will be specified as:

CT.name=Name
CT.value=Value

And similar approach will be followed when there are more than two attributes for the CT.

If the complex datatype has an annotation XSD element [1] then an extra child will be added to *CT* that will include the annotation element name and its value.

Definition 2: (Web Service Simple Datatype) Web services simple datatype is specified in the same way of specifying any of the *EiS* in Definition 1 because the complex datatype is itself a group of simple datatypes.

Web service simple datatype tree is similar to the tree in Figure 2 with *n=1* where *n* represents the number of children of *CT*.

The information in Definition 1 should also be transformed again to an XML document in order to comply with all the Web services standards. However, the new XML datatype description should be more understandable than the previous WSDL based descriptions.

Transformation rules have been developed to transform the complex datatype tree in Figure 2 to its counterpart XML specifications; the resulted XML document will have the structure specified in Figure 3.

```
<ComplexType name="CT.name-">
 <E1.attribute type=E1.value>
    <constraints>
     <E1.C1.attribute>E1.C1.value </E1.C1.attribute>
     <E1.C2.attribute>E1.C2.value </E1.C2.attribute>
      ...
    <E1.Cj.attribute>E1.Cj.value </E1.Cj.attribute>
   </constraints>
 </E1.attribute>
 <E2.attribute type=E2.value>
   <constraints>
    <E2.C1.attribute>E2.C1.value </E2.C1.attribute>
    <E2.C2.attribute>E2.C2.value </E2.C2.attribute>
     ...
    <E2.Ck.attribute>E2.Ck.value </E2.Ck.attribute>
    </constraints>
   </E2.attribute>
   ...
  <En.attribute type=En.value>
    <constraints>
     <En.C1.attribute>En.C1.value </En.C1.attribute>
     <En.C2.attribute>En.C2.value </En.C2.attribute>
      ...
     <En.Cm.attribute>En.Cm.value</En.Cm.attribute>
    </constraints>
   </En.attribute>
  </ComplexType>
```

Figure 3: XML description of complex datatype tree in Definition 1

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

The transformation rules from the tree model in Figure 2 of a complex datatype to the XML description of this complex datatype in Figure 3 are as follows:

- The root node CT will be transformed to a complexType element with name attribute and its value is CT.name.
- Every Node Ei to En of the root CT, will be transformed to sub-element of CT where the name of this sub-element is Ei.attribute. Each of the previous sub-elements will have a type attribute and its value is Ei.value.
- Every Ei sub-element of CT will have a sub-element called constraints.
- Every constraints sub-element of Ei will have j sub-elements where j is the number of constraining facets for the current Ei.
- Each of the constraints sub-elements will be called Ei.Ci.attribute and the value of the each sub-element will be Ei.Ci.value.

The advantages of the XML datatype specification in Figure 3 is that it is being conformed to the tree structure in Figure 2 and it is a more understandable form of the WSDL's datatype descriptions produced by Web services development techniques.

Another important advantage of the tree model in Figure 2 and the corresponding XML based model in Figure 3 is that they are independent of the Web services development technique used to build a WSDL document; this means that different development techniques' datatypes descriptions will be mapped to the same model.

5.2 Distinguishing Vague, Custom, Identical and Inconsistent Datatypes

The vague, custom, identical and inconsistent datatypes (will be called vague datatypes henceforth for short) descriptions must be distinguished by the resulted model in order to:

- Enable service providers to comment or annotate such datatypes to make it more understandable to service requesters.
- Enable service requester to choose among Web services with the same provided functionality by favoring the services with less vague datatype descriptions.

The vague datatypes list in this research is defined as:

VagueList = {anyType; arrayof+datatype; arrayOfKeyValueOf + datatype; Arrayof_XS_anyType; ArrayOfAnyTyp; char, dictionary, timeZone, timeZoneInfo, simpleTimeZone, duration, dictionary; FinalClass}

The *VagueList* is a collection of vague datatype specifications that was concluded after experimenting prototype Web services with operations that accepts or returns a sample of 30 .NET and Java datatypes, using the different Web services development techniques and platforms in Section 2.2.

For each element E_i in the tree model in Figure 2, where $i=1..n$, if this element has a constraining facet *Cj* and: ($E_i.C_j$.attribute = type and $E_i.C_j$.value belongs to VagueList) then the edge (CT, E_i) in Figure 2 will be drawn in red color in order to distinguish a specific element as having a vague datatype description.

In Figure 3, if any of the E_i sub-elements of *CT* has a type attribute with value that belongs to the *VagueList* then this sub-element will be changed to red color in order to be distinguished by service providers and requesters.

It was concluded based on the earlier experiments that the *maxOccurs* constraining facet leads to the vague and similar descriptions problems when it has the value of *unbounded* and the reason for this is that more than one collection datatypes are described using this constraining facet similarly as described in Section 4.4. To solve this problem, the following rule has been added to the rules of locating vague datatype descriptions:

(If $E_i.C_j$.attribute = maxOccurs and $E_i.C_j$.value=unbouded then this is Vague description)

The above rules will be used in the following case study to detect vague datatype descriptions.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 11 / ISSUE 2 / MAY 2016 / 64-82

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

6 Case Study

The aim of the following case study is two folded; firstly, to demonstrate the problems discussed in section 4 of producing vague, custom, inconsistent and identical datatype descriptions by different web services development techniques and tools (Section 6.1) and secondly to prove the usefulness of the proposed tree and XML models, introduced in Section 5, in solving the WSDL's datatype descriptions problems (Section 6.2).

6.1 The Problem

A prototype Web service with only one operation has been implemented using both WCF and Apache Axis2. The header of the operation is described in Figure 4.

```
int[] Example(List<int> input);
```

Figure 4: The header of prototype provider side Web service operation

The datatype description inside WSDL generated by WCF corresponding to the operation in Figure 4 is shown in Figure 5 while the datatype specification generated by Axis2 for the same operation is shown in Figure 6.

```
<xs:element name="Example">
 <xs:complexType>
   <xs:sequence>
<xs:element xmlns:q1="http://schemas.microsoft.com/2003/10/Serialization/Arrays"
 minOccurs="0" name="input" nillable="true"
type="q1:ArrayOfint"/>
   </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="ExampleResponse">
 <xs:complexType>
  <xs:sequence>
<xs:element xmlns:q2="http://schemas.microsoft.com/2003/10/Serialization/Arrays"
 minOccurs="0" name="ExampleResult"
nillable="true" type="q2:ArrayOfint"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Figure 5: *WCF* generated datatype specifications for the datatypes in Figure 4

```
<xs:complexType name="Example">
 <xs:sequence>
  <xs:element name="arg0" type="xs:int"
   minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
<xs:complexType name="ExampleResponse">
 <xs:sequence>
  <xs:element name="return" type="xs:int" nillable="true"
      minOccurs="0 "maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
```

Figure 6: *Apache Axis2* generated datatype specifications for the datatypes in Figure 4

The following observations can be concluded from Figure 5 and Figure 6:

- It will be difficult for a service requester to understand that the required provider side operation (Figure 4) has a list input based on the WCF generated datatype description in Figure 5 or the Axis2 generated datatype description in Figure 6. This indicates that these datatype descriptions are vague as described in Section 4.1.

- WCF used a custom datatype (Section 4.2) *ArrayOfin* to describe the provider side array and list datatypes in Figure 4 whereas Axis2 used only XS datatypes and made use of the minOccurs and maxOccurs constraining facets to describe both of the array and list datatypes.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 11 / ISSUE 2 / MAY 2016 / 64-82

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

- Both WCF and Axis2 specified the list datatype in Figure 4 in the same way that is used to specify arrays because both the input and the output of the operation in Figure 4 were specified similarly (using *ArrayOfInt* in case of WCF and minOccurs, maxOccurs constraining facets in case of Axis2) despite the fact that the input in Figure 4 of the provider side operation is a list and the output is an array. This is an example of using identical descriptions in Section 4.3.

- There is inconsistency in specifying the same method in Figure 4 by WCF (Figure 5) and Axis2 (Figure 6). This is an example of the inconsistency problem in section 4.4.

So, it can be concluded that the four problems of datatype description mention in Section 4 do exist in the case study described by Figure 4, 5, and 6.

6.2 The Tree and XML based Datatype Models

Figure 7 and Figure 8 below represent the trees corresponding to the complex datatypes in Figure 6 (produced by *Axis2*) based on Definition 1.

In Figure 7, the complex datatype *Example* which is the root of the tree has one child called *E1* where:

E_1.attribute = arg_0
E_1.value = xs:int

E1 has two children C1 and C2 where:

$E_1.C_1$.attribute = minOccurs
$E_1.C_1$.value = 0
$E_1.C_2$.attribute = maxOccurs
$E_1.C_2$.value = unbounded

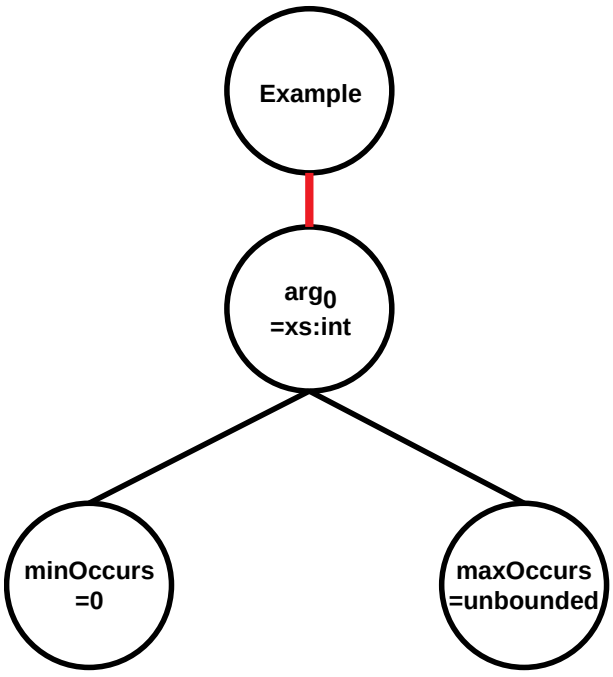

Figure 7: Tree of the complex datatype Example in Figure 6

In this example, the complex datatype Example has only one sub-element (node), however, for complex datatypes with more than one sub-element; it will be specified in similar way of *E1*. Based on the transformation rules specified by Figure 3, the XML specifications corresponding to the tree models in Figure 7 and Figure 8 is described by Figure 9 below.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 11 / ISSUE 2 / MAY 2016 / 64-82

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

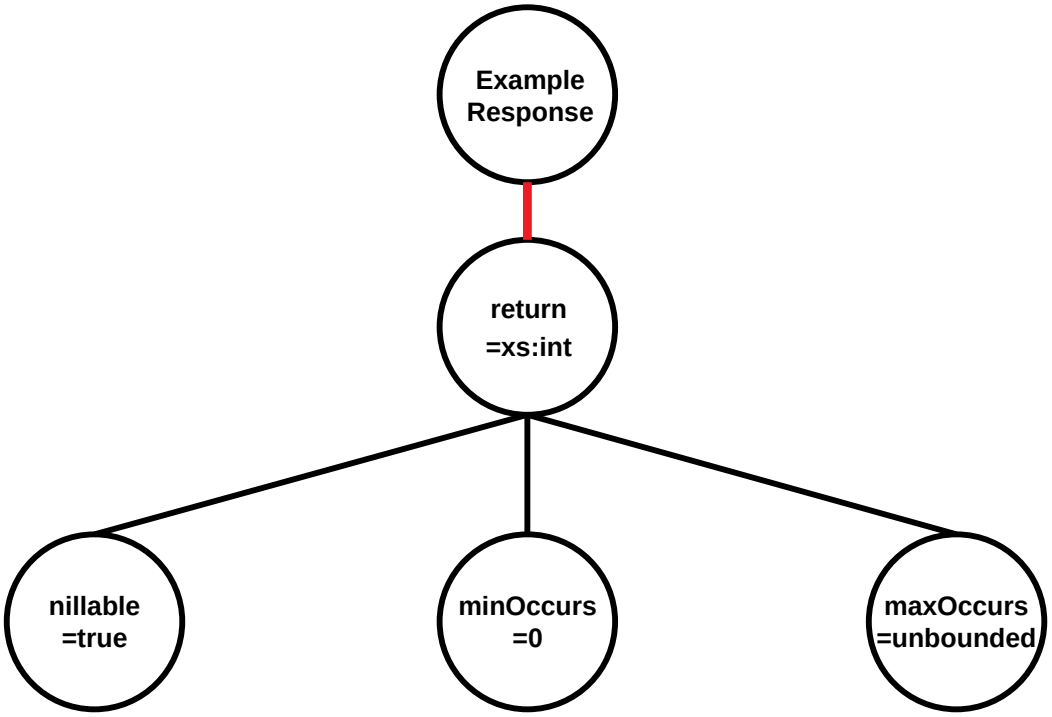

Figure 8: Tree of the complex datatype ExampleResponse in Figure 6

```
<ComplexType name="Example">
  <arg0 type=xs:int>
   <constraints>
         <minOccurs>0</minOccurs>
         <maxOccurs>unbounded</maxOccurs>
   </constraints>
  </arg0>
</ComplexType>
<ComplexType name="ExampleResponse">
   <return type=xs:int>
   <constraints>
        <nillable>true</nillable>
         <minOccurs>0</minOccurs>
         <maxOccurs>unbounded</maxOccurs>
    </constraints>
   </return>
 </ComplexType>
```

Figure 9: XML based datatype model of Figure 2 and 3 based on the Tree model in Definition 1

The following elaborations must be made on the tree and XML models in Figure 7, Figure 8, and Figure 9:

- In Figure 7, the edge between the *Example* root node and *arg0* was red to indicate that *arg0* corresponds to a vague datatype description, this conclusion was made based on the rule in Section 5 that considers the *maxOccurs=unbounded* as vague datatype description because it will be difficult to be understood by service requesters. For a similar argument, the edge in Figure 8 between *ExampleResponse* and *return* nodes was red too.

- In Figure 7 and 8, the root of the tree has only one child because the original complex datatypes *Example* and *ExampleResponse* in Figure 6 has only one element each; namely, *arg0* element for *Example* and *return* element for *ExampleResponse*. If a complex datatype has more than one element then the corresponding tree will have more than one child depending on the number of elements. The edge between the root and each child will be red only if the child represents a vague datatype description according to the discussion made in Section 5.

- When transforming the tree in Figure 7 and 8 to their equivalent XML description, *arg0* and *return* elements were in red because the edges corresponding to these elements were red.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

- The tree based models and the corresponding XML description has been described only for the WSDL produced using the apache Axis2, however, similar approach can be applied to reach for the similar models in case of the WSDL produced by WCF for the same case study example in Figure 4.

The benefits of the tree models and the XML description are:

- These models are easier to be understood by both service requesters and providers of Web services and this fact will also be proved using the questionnaire is Section 7.2.

- The red edges in the tree models and the corresponding red elements in the XML descriptions will be an indication to the service provider of the described Web service to add the suitable comments or annotation for the vague datatype descriptions in the original WSDL document in order to enhance the described Web service's understandability and discoverability.

- Web service requesters can benefit from the models to avoid the Web service with many vague descriptions by using another Web service with similar functionality by less vague descriptions.

- In a summary, it can be concluded from the case study that the tree and XML models are useful in detecting vague datatype descriptions inside WSDL in a way that enables web services providers to find a mean to annotate these descriptions to reach for a more understandable and reusable Web services.

- The limitations of the proposed approach is that it only considers the Web services development techniques described in Section 2.2 and Section 4; however, Web services builders or developers might use other development techniques.

7 Implementation and Evaluation

A proof of concept tool that models Web services datatype descriptions based on the analysis in Section 5 has been implemented as a Web application using Active Server Pages (ASP.NET) framework.

7.1 Implementation

The tool implements the following activities:

- Activity 1: The tool allows the user, who could be a developer that builds Web services, or a developer that consumes Web services in his applications, to insert the URL of the WSDL document to be investigated by the tool.

- Activity 2: The tool parses the WSDL document inserted in Activity 1 and locates the datatypes section inside WSDL which is called *types* element by WSDL. The *types* element could be either inside WSDL itself or in a separate XS definitions document with a location specified inside WSDL.

- Activity 3: Each complex and simple element inside the types element will be mapped to a tree model as described in Definition 1, Definition 2, and Figure 2.

- Activity 4: The procedure of distinguishing vague datatype descriptions in Section 5.2 will be applied for each complex and simple datatype and accordingly the resulted models in Activity 3 will be modified to show the vague elements in red color.

After distinguishing vague datatype description, service provider can go back to the WSDL document to comment or annotate these datatypes in order to reach for a more understandable WSDL document.

7.2 Evaluation

To evaluate the usefulness of the prototype proof of concept tool, it was used by 20 service providers (developers that build Web services) and 12 service requesters (developers that consume Web services in their Web applications). Both of the Web services providers and requester use WSDL documents extensively in their daily work. The 32 Web services builders and consumers were chosen from 5 different software companies.

An experimental Web service with operations that accept and return a sample of 30 *Java* datatypes (Table 1) was built using both *Java EE 6* platform and the same Web service was built also with apache *Axis2*. Similar Web service was built again but using a sample of equivalent *.NET* datatypes (Table 1) and *WCF* framework. The resulted 3

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 11 / ISSUE 2 / MAY 2016 / 64-82

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

WSDL documents were then fed to the tool mentioned in Section 7.1 in order to model the datatype specifications of each WSDL.

Table 1: Experimental *Java* and *C#* datatypes

datatype Category	**Java**	**.NET**
Primitive datatypes	byte, int, BigInteger, short, long, float, double, char, boolean, BigDecimal, string, Object	byte, sbyte, int, short, long, float, double, char, bool, decimal, string, object
Collection datatypes	Array, List, ArrayList, LinkedList, Map, Set, HashTable	Array, List, ArrayList, SortedList, LinkedList, HashTable, Dictionary
Dates and Times datatypes	Date, Calendar, GregorianCalendar, TimeZone, simpleTimeZone	DateTime, DayOfWeek, TimeSpan, TimeZone, TimeZoneInfo
Enumeration	Enum	enum
User defined class	Class	class
Structure or final class	final class	struct
Abstract class	abstract class	abstract class
Interface	Interface	Interface
Exception classes	Throwable	Exception

The 2 auto-generated WSDL documents by the *Java* based platforms (*Java EE 6 and Apache Axis 2*) were handled to the Web services providers and requesters that use the *.NET WCF* framework to build their applications. And vice versa; the auto-generated WSDL document by the *WCF* framework was handled to the Web services providers and requesters who use the *Java* based platforms to build their applications. The reason after that was to assess the understandability of WSDLs when these WSDLs are produced by different platforms than ones used by providers and requesters.

After that; both providers and requesters were asked to fill the questionnaire in Appendix A that contained questions about how useful the tool is in their opinion, and also how they think the tool can be improved.

The following results had been concluded based on the questionnaire:

1. Both Web services providers and requesters faced difficulties to understand the described Web services based only on WSDL. This fact was concluded because they needed, on the average, 15 minutes to answer the questions in task 1 of the questionnaire.

2. The major concern of the service providers and requesters who use the Java based platform is that the .NET WCF based WSDL has lots of custom .NET datatypes. While the major concern of the service providers and requesters that use the .NET WCF framework is that the Java based platforms is not understandable because it depend on cardinality attribute only to specify collect datatypes. These problems were explained in Section 6 in details.

3. The answers for question 1 in Task 2 are analyzed in Table 2 below:

Table 2: Answers of web services users about the benefit of the datatypes models

Benefit	**Number**	**Percent**
5	19	59.4%
4	11	34.4%
3	2	6.2%
2	0	0%
1	0	0%

In table 2, about 90% of the Web services users gave a score of 5 and 4 to the usefulness of the proposed models and this is a great indication to the role that these models can do to improve Web services datatype descriptions' understandability.

4. The main suggested improvement by service providers and requesters was to enhance the tool to make it consider more datatypes and more Web service development techniques and platforms. Also 5 Web services builders suggested that the tool should provide a semantic annotation for the vague datatype specifications; however, they did not mention how such annotations can be reached.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

8 Conclusion and Future Work

Web services are important for developing and integrating Web based applications including e-Commerce applications. Web services datatype descriptions still face many problems such as producing vague, custom, identical, and inconsistent datatype descriptions inside WSDLs generated by different Web services development techniques.

Web services community must find solutions to the Web services datatype descriptions problems in order to increase Web services interoperability and understandability by service requesters and providers. Improving interoperability and understandability and other quality attributes of Web services will lead eventually to increasing the usage of Web services in Web applications development.

This research introduces an approach to solve the problem of Web services datatype descriptions by modelling these descriptions in a way that reflects the problems of vague, custom, identical, and inconsistent datatype descriptions discussed in Section 4.

Using the proof of concept tool in Section 7, the research's approach proved to be useful for both Web services providers and requesters in understanding the datatype descriptions inside WSDLs.

Future work will consider improving the approach in this paper of modeling datatype descriptions by considering more Web services development techniques, tools, and platforms. The future work will also consider finding an approach to enrich WSDL's datatype descriptions that have problems similar to the problems in Section 4. The enrichment can be based on using the semantic based approaches such as those described in [18].

Acknowledgments

This research was fully supported by a research grant from Philadelphia University in Jordan. Special thanks to the editor and the reviewers for their invaluable remarks that contributed to the improvement of this research.

Websites List

Site 1: Using W3C XML Schema
http://www.xml.com/lpt/a/691

References

[1] G. Alférez, R. M. V. Pelechano, C. Salinesi and D. Diaz, Dynamic adaptation of service compositions with variability models, The Journal of Systems and Software, vol. 91, no. 2014, pp. 24-47, 2014.
[2] A. Bellini, P. A., L. Aparecida and M. Zaina, Top-Down Approach for Web Services Development, in Fifth International Conference on Internet and Web Applications and Services (ICIW), Spain, 2010, pp. 426-431.
[3] M. B. Blake and M. F. Nowlan, Taming Web Services from the wild, IEEE Internet Computing, vol. 12, no. 5, pp. 62-69, 2008.
[4] R. Chinnici, R. Moreau, A. Ryman and S. Weerawarana, Web Service Description Language (WSDL) Version 2.0, Part 1: Core Language, W3C Recommendation 26 June 2007, W3C. [Online]. Available: https://www.w3.org/TR/wsdl20/.
[5] P. Cibraro, K. Claeys, F. Cozzolino and J. Grabner, Professional WCF 4 Windows Communication Foundation with .NET 4, Wiley Publishing Inc, 2010.
[6] J. L. O. Coscia, C. Mateos, M. Crasso and A. Zunini, Refactoring code-first Web Services for early avoiding WSDL anti-pattern: Approach and comprehensive assessment, Science of Computer Programming, vol. 89, no. 2014, pp. 374-407, 2014.
[7] M. Crasso, J.M. Rodriguez, A. Zunino and M. Campo, Revising WSDL Documents: Why and How, IEEE Internet Computing, vol. 14, pp. 48-56, 2010.
[8] K. Dezhgosha and A. Swathi, Web services for designing small-scale Web applications, in IEEE International Conference on Electro Information Technology, Lincoln, Nebraska, USA, 2005, pp. 1-4.
[9] H. El Bouhissi and M. Malki, Reverse Engineering Existing Web Services Applications, in IEEE 16th Working Conference on Reverse Engineering (WCRE), Lille, France, 2009, pp. 279-283.
[10] Z. Fang, L. Liao and R. Chen, Automatic Verification of Data Centric Web Services Specifications, IERI Procedia, special issue about the International Conference on Electronic Engineering and Computer Science (EECS 2013), vol. 4, no. 2013, pp. 93-98, 2013.
[11] S. Graham, D. Davis, S. Simenov, G. Daniels, P. Brittenham, Y. Nakamura, P. Fremantle, D. konig and C. Zentner, Building Web Services with Java, Making sense of XML, SOAP, WSDL, and UDDI, 2nd ed., Sams Publishing, 2005.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

[12] R. Gronmo, D. Skogan, I. Solheim and J. Oldevik, Model-Driven Web Services Development, International Journal of web Services Research (IJWSR), vol. 1, no. 4, pp. 1-13, 2004.
[13] D. Jayasinghe and A. Azeez, Apache Axis2 Web Services, Packt publishing, 2011.
[14] J. Jiang and T. Systa, UML-Based Modeling and Validity Checking of Web Services Descriptions, in IEEE International Conference on Web Services (ICWS'05), Florida, USA, 2005, pp. 453-460.
[15] S. N. kumar, P. Pabitha and A. M. Ahamed, Web Service Discovery based on Semantic Description, in International Conference on Cloud & Ubiquitous Computing & Emerging Technologies, Pune, India, Nov. 2013, pp. 199-203.
[16] C.-H. Lee and S.-Y. Hwang, A Model for Web Services Data in Support of Web Services Composition and Evaluation, in World Conference on Services – I, California, USA, 2009, pp. 384-391.
[17] A. d. Melo and P. Silveira, Improving data perturbation testing techniques for Web services, Information Sciences, vol. 181, no. 3, pp. 600-619, 2011.
[18] H. Nacer and D. Aissani, Semantic web services: Standards, applications, challenges, Journal of Network and Computer Applications, vol. 44, pp. 134-151, 2014.
[19] J. Pasley, Avoid XML schema wildcards for Web Service interfaces, IEEE Internet Computing, vol. 10, no. 2006, p. 72–79, 2006.
[20] D. Paulraj, S. Swamynathan and M. Madhaiyan, Process Model Ontology-Based Matchmaking of Semantic Web Services, International Journal of Cooperative Information Systems, vol. 20, no. 4, pp. 357-370, 2011.
[21] D. Peterson, S. Gao, A. Malhotra, C. M. Sperberg-McQueen and H. Thompson, "W3C XML Schema W3C XML Schema W3C XML Schema Definition Language (XS) 1.1 Part 2: datatypes, W3C Proposed Recommendation 5 April 2012," W3C. [Online]. Available: https://www.w3.org/TR/xmlschema11-2/.
[22] G. Rai, G. R. Gangadharan and V. Padmanabhan, Algebraic modeling and verification of Web service composition, Procedia Computer Science, special issue 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015), vol. 52, no. 2015, pp. 675-679, 2015.
[23] J. M. Rodriguez, M. Crasso, A. Zunino and M. Campo, Improving Web service descriptions for effective service discovery, Science of Computer Programming, vol. 75, no. 11, pp. 1001-1021, 2010.
[24] M. Á. Rodríguez-García, R. Valencia-García, F. García-Sánchez and J. J. Samper-Zapater, Ontology-based annotation and retrieval of services in the cloud, Knowledge-Based Systems, vol. 56, no. 2014, pp. 15-25, 2014.
[25] M. Sabou and J. Pan, Towards semantically enhanced Web services repositories, Web Semantics, Services and Agents on the World Wide Web, vol. 5, no. 2, pp. 142-150, 2007.
[26] Q. Sheng, Z. Maamar, H. Yahyaoui, J. Bentahar and K. Boukadi, Separating Operational and Control Behaviors: A New Approach to Web Services Modeling, IEEE Internet Computing, vol. 14, no. 3, pp. 68-76, 2010.
[27] W. Sun, S. Li, D. Zhang and Y. Yan, A Model-driven Reverse Engineering Approach for Semantic Web Services Composition, in IEEE World Congress on Software Engineering (WCSE), China, 2009. pp. 101-105.
[28] C. Taconet and Z. Kazi-Aoul, Context-awareness and Model Driven Engineering: Illustration by an E-commerce application scenario, in Third International Conference on Digital Information Management. ICDIM 2008, pp. 864-869.
[29] H. N. Talantikite, D. Aissani and N. Boudjlida, Semantic annotation for web services discovery and composition, Computer Standards & Interfaces, vol. 31, no. 6, pp. 1108-1117, 2009.
[30] D. Tosi and S. Morasca, Supporting the semi-automatic semantic annotation of web services: A systematic literature review, Information and Software Technology, vol. 61, pp. 16-32, 2015.
[31] F. Truyen, "The Fast Guide to Model Driven, The Basics of Model Driven Architecture," OMG Whitepaper, 2006.
[32] R. D. Virgilio, Meta-modeling of Semantic Web Services, in IEEE International Conference on Services Computing (SCC), FL, USA, 2010, pp. 162-169
[33] W. Yan-xin, L. Xiang-yabg and L. Ying-xiong, Web-Services Based Model Management in E-commerce Recommender Systems, in 19th International Conference on Management Science & Engineering, Dallas, USA, 2012, pp. 3-38.
[34] J. Yu, Q. Z. Sheng, J. K. Swee, J. Han, C. Liu and T. H. Noor, Model-driven development of adaptive web service processes with aspects and rules, Journal of Computer and System Sciences, vol. 81, no. 3, pp. 533-552, 2015.
[35] X. Zhang and H. Liu, A Context-aware Web Service Composition Model for Dynamic E-commerce Applications, in International Conference of Soft Computing and Pattern Recognition (SoCPaR), Dalian, China, October 14-16, 2011, pp. 364-369.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 11 / ISSUE 2 / MAY 2016 / 64-82

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762016000200006

Appendix A: WSDL Understandability Questionnaire

Task 1: Based on the attached WSDL document describe:

1. The operations provided by the described Web service.

2. The datatypes of each operation.

3. What are, in your opinion, the operations inside WSDL that are difficult to be invoked due to lack of understandability of its input or output datatype descriptions.

After that the tree and XML based models corresponding to the datatypes in the WSDL investigated by each Web service provider or requester were given to him and after that he was asked to accomplish task 2 below.

Task 2: Check the attached tree and XML models that describe the datatypes described by the WSDL you have investigated in Task 1 and then answer the following:

1. Have the models helped you to understand the datatypes of the described Web service more than WSDL? Give mark from 1 to 5 where 5 means the models help you a lot and 1 means you did not get any benefit from the models.

2. How do you think these models can be improved?