www.ietdl.org

Published in IET Software
Received on 21st July 2010
Revised on 19th November 2010
doi: 10.1049/iet-sen.2010.0089

ISSN 1751-8806

# Metrics suite for maintainability of eXtensible Markup Language web services

*D. Baski*[1] *S. Misra*[2]

[1]*BILGI Geographic Information Conversion & Management System Co. Ltd, Ankara, Turkey*
[2]*Department of Computer Engineering, Atilim University, Ankara*
*E-mail: ssopam@gmail.com*

**Abstract:** The eXtensible Markup Language (XML) web services are emerging as the de-facto mechanism for exchanging structured information between applications. The large popularity and acceptance of web services led the developers to adopt the best practices of web service implementation and to find the ways for managing and maintaining web services more effectively. Maintainability, one of the important factors, which affects the quality of XML web services, can be controlled by the proper software metrics that are specifically designed and developed for it. In this paper, we present a suite of metrics to evaluate the quality of the XML web service in terms of its maintainability. The present suite of metrics includes: data weight of a web service description language, distinct message ratio metric, message entropy metric and message repetition scale metric. All the proposed metrics have been evaluated theoretically and validated empirically. A comparative study with similar measures proves the worth of the metric suite.

## 1 Introduction

As the internet technologies have been evolving software solutions have required distributed computing where diverse applications run on different platforms needed to communicate and exchange application data. Towards the end of 1990s organizations have turned their attentions to web applications. With the emergence of web applications the idea of integrating them as a very loosely coupled software components leads to the development of web services, [1–5] whose implementation details are hidden behind their interfaces. web services that are based on eXtensible Markup Language (XML) technologies enable integration of diverse IT processes and systems and have been gaining extraordinary acceptance from the basic to the most complicated business and scientific processes. The developers can build new, more powerful applications by exposing existing applications as XML web services and using it as building blocks. Particularly for the business people web services have become a powerful means to achieve their business goals. By using open technology (XML and the internet protocols) and open standards managed by broad consortia such as, open standards for information society (OASIS) [6–9] and the W3C integrating application functionality within an organisation or integrating applications between business partners is achieved more rapidly, easily and cheaply than ever before. Owing to the fact that the XML web services are based on an open standardised suite of technologies such as XML [10], hyper text transport protocol (HTTP) [2], simple object access protocol [1–5, 11] (SOAP), universal description, discovery and integration [12] (UDDI) and web service description language [1–5, 13] (WSDL) industry-wide support graded the popularity and importance of this platform.

In the software development life cycle, software metrics play an important role since they provide useful feedback to the designers as to the impact of decisions that are made during design, coding, architecture or specification phases; without such feedback, many decisions are made in *ad hoc* manner. Many software metrics have been proposed for decades to improve the quality of the traditional software products; unfortunately, not too much effort have been made to develop the metrics for web services. The lack of research in developing software metrics for the web services is our main motivation to find useful metrics for the assessment of the quality of the web services in terms of maintainability. The maintainability is one of the important factors that affect the quality of the web services that can be seen as a kind of software project. Since the degree in complexity has effect on maintaining any kind of software project, we have studied in developing metrics to measure the complexity of a web service. The complexity degree of a web service can be measured by analysing its WSDL document because WSDL provides the description of the web service to the service requestors. However, the WSDL does not contain information about implementation details of a web service, hence, we can only measure the data complexity of a web service. The data complexity of a web service can be defined as the complexity of data flowed to and from the interfaces of a web service and can be characterised by an effort required to understand the structures of the messages that are responsible for exchanging and conveying the data. In this point of view, it will help to analyse the structures of the messages for measuring the data complexity of a web service via the usage of software metrics.

In this paper, we propose a suite of metrics to evaluate and maintain the quality of the web service in terms of its maintainability. In the suite of metrics, we have already presented one metric: data weight of a WSDL [DW (WSDL)] as a preliminary work [14]. However, to make the paper self-contained, we provide the full definition of DW (WSDL). Further, we have extended our previous work by including three more metrics, and presenting a suite of metrics for web services. All the presented metrics have been evaluated theoretically and validated empirically by real examples.

This paper is organised as follows. In Section 2, a brief overview of the architecture of web services and the structure of WSDL documents are given. Researches related to WSDL metrics are reviewed in Section 3. The proposed metrics suite is demonstrated by examples in Section 4. Empirical validation and a comparative study with other measures have been done in the same section. The theoretical validation of proposed metrics is given in Section 5. The summary and short analysis of the work are given in Section 6. Lastly Section 7 provides the concluding remarks and future works.

## 2 Background: XML web services

There are several definitions developed for a web service. The web services architecture working group of the W3C [15] declared the following definition for a web service:

> 'A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards'.

In the web services architecture, the service provider creates a web service and defines offered service(s) by using WSDL documents and then publishes the service via the service registry (UDDI) to make the offered services available for the service requesters to easily access. Once a web service is published, a service requester may find the needed service via the UDDI interface. The UDDI registry provides the service requester with a WSDL service description and a uniform resource locator pointing to the service itself. Once service consumers have the WSDL file, they can find how to communicate with the web services by using the proper transport protocols such as SOAP. The service requester may then use this information to directly bind to the service and invoke it.

If we think web services as remote objects that can be exposed through WSDL, the SOAP provides a mechanism to remotely access to these objects across the internet without having problems with regard to the integration and interoperability of different enterprises. In this mechanism, XML documents are used for representing and transporting data to and from integrated applications' public interfaces.

In order for this mechanism to work, W3C describes a framework for web services consisting of a foundation built on top of three core XML specifications:

1. Simple object access protocol (SOAP),
2. Universal description, discovery, and integration (UDDI) and
3. Web services description language (WSDL).

The following subsections provide an overview for each specification.

### 2.1 Simple object access protocol-SOAP

SOAP is the communication protocol intended for exchanging structured information in a decentralised, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics [16].

### 2.2 Universal description, discovery and integration

The specifications of UDDI [12] sponsored by OASIS [6] define a registry service for web services and for other electronic and non-electronic services [7–9]. A UDDI registry service is itself a web service that manages information about service providers, service implementations and service metadata. While service providers use UDDI to advertise the services they offer the service requesters use UDDI to discover services that meet their needs and to obtain the service metadata needed to consume those services.

### 2.3 Web service description language

WSDL, a W3C recommendation, provides a model and an XML format for describing and locating web services. The first version of WSDL 1.1 [13] was submitted to the W3C in late 2001. Later, W3C web service description working group was formed in early 2002 to standardise WSDL and as a result the standard version WSDL 2.0 is produced that includes significant changes and improvements [17] in its previous version.

The most advantageous feature of WSDL is that by creating the WSDL document the owner of the web service can separate the description of the abstract functionality offered by the service from concrete details of a service description. In other words, WSDL provides a platform for integrating diverse applications regardless of their place, that is, where they reside and how they are implemented. Further, when integration of diverse applications is aimed, the exposed web services should be designed to have concise and clear agreement on the common specifications of protocols and data-type systems that can be used by a diverse set of programming languages to work in an interoperable manner. The format of the messages exchanged between a service provider and consumer must be well defined so that the message sender can easily construct and the message receiver can process. For the web service to be easily invoked the WSDL documents provide not only a way for grouping messages into operations and operations into interfaces but also provide a way for defining bindings for each interface and protocol combination along with the endpoint address for each one. Since underlying business and data models used by applications that are indented to be integrated may change over time, for accommodation of these changes building a flexible document structure that can be extended will pay off in the future. In [18], the use of WSDL and XML schema is mandated for describing web services. In this way the interoperability at the service description layer is ensured. In fact, to provide a common data-type system and to represent application data the WSDL document is supported by XML Schema that is either imported to or embedded into the WSDL document and encapsulated by <types> construct

of WSDL. It is worth mentioning here that there are several languages [10, 19–24] that produce XML schema documents. Among them, W3C XML Schema and DTDs are the most favoured schema languages for generating XML documents. Further, the decision in XML schema design to represent the application's data exchanged may affect the understandability of the exchanged messages described in the WSDL document and in turn comprehending of a web service defined by the WSDL document.

The WSDL consists of several elements: Type, operations, messages, portType, binding, port and service. WSDL Type's elements are described as a container for abstract-type definitions defined using XML Schema. The Type definitions are referenced from higher-level message definitions to define the structural details of the message. Although WSDL 1.1 [13] allows the use of any type definition language, it strongly encourages the use of XML schema. An operation is the description of an action. A WSDL document describes a web service as a collection of abstract items called ports or endpoints. The WSDL message element defines an abstract message that can serve as the input or output of an operation. Messages consist of one or more part elements, where each part is associated with either an element or a type. The WSDL portType element defines a group of operations, also known as an interface. A portType element contains zero or more operation elements. The WSDL binding document defines the protocol details for operations and messages defined by a particular portType. It also describes the concrete details of using a particular portType with a given protocol. The WSDL service element defines a collection of ports, or endpoints, which expose a particular binding.

## 3 Related work

As we mentioned earlier that not too much work have been done to propose metrics for web service except for a few. Yu *et al.* [25] studied to compare web services with other traditional software components by adopting some existing metrics to WSDL documents that were developed for measuring the interface complexities of software components. Some of the metrics presented in [25] are reviewed below.

*Argument per operation (APO):* Given the total number of arguments ($n_a$) in total number of operations ($n_o$) described in WSDL is used for measuring the size of Web service and is defined as

$$\text{APO} = \frac{n_a}{n_o} \tag{1}$$

*Distinct argument ratio (DAR):* The ratio of distinct (argument, type) pairs appeared in a WSDL interface description is defined in terms of distinct argument count (DAC) and given by

$$\text{DAR} = \frac{\text{DAC}}{n_a} \tag{2}$$

*Argument repetition scale (ARS):* Given a sequence A of (name, type) pairs in the web service, the (ARS) is

$$\text{ARS} = \frac{\sum_{a \in A} |a|^2}{|A|} \tag{3}$$

where $|a|$ denotes the number of repetition of $a$ in $A$.

In addition to these metrics that are originally developed for measuring interface complexity of traditional software components, some OO-based metrics listed below were also adapted to WSDL documents for measuring complexity of web service.

*Operations per service (OPS):* Similar to the OO metric weight method per class operation per service is defined as the total count of operations that are declared by a PortType of a web service and intended to measure the size of a given WSDL document.

*Depth of inheritance tree (DIT):* DIT is the length of the inheritance path in the directed acyclic graph (DAG) of XSD that shows the inheritance relation of the type definitions for the arguments.

*Number of children (NOC):* NOC is the immediate NOC of a node in the XML Schema inheritance tree.

### 3.1 Motivation and discussion

According to Yu *et al.* [25], while lower ARS indicates more specialised functionality of a web service, higher DAR indicates more consistency of argument declarations.

Consistent argument declarations make it easier to understand and reuse the components. Furthermore, they claim that having higher DAR and lower ARS indicate more reusable ability of WSDL interfaces but also more cumbersome to understand the functionality of each operation [25]. However, we observed that the meaning of higher DAR value to reflect the consistency in argument declaration is misunderstood. According to the original definition of DAR given by Boxall and Araban [26], the interfaces with lower DAC and DAR will have arguments that are declared more consistently. This interpretation is more realistic since having less number of distinct arguments implies that the arguments are more reused in the interfaces and hence results in more consistency. So, lower DAR and higher ARS indicates that a given interface is less complex in terms of understandability of its functionality.

We also observed that, on the other hand, it is not always the case where having higher OPS value for a given web service indicates that the web service under consideration is more complex than that of service having lower OPS value. Consider, for example, two WSDL documents shown in

```
...
<wsdl:types>
  <s:schema targetNamespace="exampleSchema/" xmlns:tns=" exampleSchema/">
<s:simpleType name="string1">
  <s:restriction base="s:string">
    <s:enumeration value="a" />
    <s:enumeration value="b" />
  </s:restriction>
</s:simpleType>
<s:simpleType name="list"/>
  <s:list itemType="s:int"/>
  </s:simpleType>
<s:simpleType name="union">
  <s:union memberTypes="s:date s:string"/>
  </s:simpleType>
....
</wsdl:types>
<message name="M1">
        <part name="arg0" type="tns:string1" />
        <part name="arg1" type="tns:list" />
</message>
<message name="M2">
        <part name="arg2" type="tns:union" />
</message>
...
<portType name="X">
            <operation name="Y">
            <input message="tns:M1" />
            <output message="tns:M2" />
        </operation>
...
```

**Fig. 1** *Example WSDL with data type as a simple type*




doi: 10.1049/iet-sen.2010.0089

```
.....<wsdl:types>
<s:schematargetNamespace="exampleSchema2/"xmlns:tns=" exampleSchema2/">
  <s:element name="X">
   <s:complexType>
    <s:sequence>
     <s:element name="e" type="s:date"/>
    </s:sequence>
   </s:complexType >
  </s:element>
  <s:element name="Y">
     <s:complexType>
      <s:complexContent>
       <s:restriction base="s:anyType">
        <s:sequence>
          <s:element name="c" type="s:string" />
          <s:element name="d" type="s:string" />
        </s:sequence>
       </s:restriction>
      </s:complexContent>
</s:complexType>
</s:element>
....
</wsdl:types>
<wsdl:message name="M1">
   <wsdl:part name="parameters" element="tns:X" />
 </wsdl:message>
 <wsdl:message name="M2">
   <wsdl:part name="parameters" element="tns:Y" />
 </wsdl:message>
.....
<wsdl:portType name="S">
  <wsdl:operation name="OP1">
     <wsdl:input message="tns:M1" />
     <wsdl:output message="tns:M2" />
  </wsdl:operation>
.....
</wsdl:portType>
.....
```

**Fig. 2** *WSDL document with data type as complex type*

Figs. 1 and 2, both offer only one operation having three arguments each has different data type.

While the first service's arguments have derived simple types, the data types of the arguments in the second service are defined via the usage of W3C XML schema's element declaration that have complex types.

As seen in Fig. 1, for the first WSDL DIT value is 1 since the data type of its arguments are defined as a simple type and one of them is derived by restriction from built-in simple type string of W3C XML schema and, NOC value is 1 since there is only one restriction according to DAG evaluation defined in [25]. The second WSDL in Fig. 2, on the other hand, also has DIT value of 1 and NOC value of 1.

Based on these two WSDL documents both services have equal APO, DAR, ARS, OPS, DIT and NOC values that are not sufficient to differentiate these two services in terms of their complexities. This is the first reason for us to develop the new metrics that can differentiate the web services in terms of the complexity and structures of the messages. We make it more clear in next paragraph.

In general, while searching an appropriate web service the main consideration of a service consumer is to inspect the list of supported operations hosted in the portType construct of WSDL to get an overall feel for what a web service offers since operations are the focal points of interacting with the service. As the execution of an operation requires exchanging the requesting and responding messages between the service requestor and the service provider, the structure of exchanged messages of the operations such as number of arguments contained, the data types of these arguments should also be considered. According to this point of view understandability of message structures has an important role in comprehending the operations that a web service offers. This is our main motivation to propose metrics and accordingly, we have developed a metrics suite in the next section.

## 4 Proposed metrics: demonstration, and validation

In the previous section, we concluded that complexity of web services depends on the operations performed by messages between the service requestor and the service provider, and therefore the structure of exchanged messages of the operations such as number of arguments contained and the data types of these arguments are considered for measuring complexity. Further, the arguments that an operation takes are contained in the message constructs of WSDL and each message construct can host one or more argument definitions. Each argument is defined by one part element of a message construct and each part element provides either element or type reference via its element or type attributes to associate the arguments with the components of the XSD that consist of element, type, definition/declarations for defining the data types of these arguments. Since, XSD is responsible for defining the data types of the arguments; the complexities of these type definitions in XSD also affect the understandability of the message structures. As defined before the data complexity of a web service refers to the effort required to understand structures of the messages that are responsible for exchanging and conveying the application data. Hence, web service's data complexity can be measured by analysing the XSD embedded in WSDL and the structures of its requesting and responding messages used by the operations that a web service offers. In order to measure the data complexity we use complexity metric *C* (XSD) [27]. The complexity of XSD document measured by *C* (XSD) is evaluated by summing up all of its components' complexities. In this evaluation each component's complexity degree is reflected by a weight value that is assigned based on the component's internal architecture.

The complexity degree of each XSD component affects the understandability of the message structures since these components are associated with the arguments for describing those arguments' data types. This point has been ignored by the metrics in [25] and becomes our main focus to measure data complexity of a web service.

By analysing the structures of the exchanged messages described in WSDL we measure the data complexity of a given web service via our proposed metrics suite introduced in the following subsections. We have also carried out empirical validation of our proposed metrics through analysing 56 different real WSDL documents written in WSDL 1.1[13]. The statistics referring these measurement results have been collected and shown in Table 3. Additionally, to verify the soundness and robustness of our metrics, each proposed metric is compared with similar metrics.

The analysed WSDL documents are collected by some well-known sites http://www.xmethods.com/, http://www.webservicelist.com/, that list publicly available web services. The references to these analysed WSDL documents are given in Table 4.

### 4.1 Data weight of a WSDL

The DW of a given WSDL can be defined as the sum of the data complexities of each input and output messages [27]. By analysing the message structures that contain the arguments that the operations of a web service take, we can measure the data complexity of each message.

The data complexity of a message can be evaluated by summing the weight values of each part elements belonging to the message construct of WSDL. Consequently, the total data complexity of the WSDL can be evaluated by summing

up the data complexities of all of its messages. Hence, we introduce DW of WSDL [DW (WSDL)] metric to capture the complexity of a web service owing to the data flow through its interfaces. In this aspect the DW metric can be useful for assessing the effort required for understanding the data types of the arguments taken by the operations that a web service provides. We can define DW metric as follows:

*Definition 1:* DW (WSDL): Given a WSDL document having $n_m$ number of messages in total, DW of WSDL metric is defined as

$$\text{DW (WSDL)} = \Sigma_{i=1}^{n_m} C(M_i) \tag{4}$$

where $n$ is the number of messages, $C(M_i)$ is the complexity value of $i$th message and can be evaluated by

$$C(M) = \Sigma_{j=0}^{n_p-1} wp_i \tag{5}$$

where $n_p$ is the total number of <part> elements encapsulated by a given message construct, and $wp_i$ is the weight value of $i$th <part> definition of the messages exchanged.

As stated earlier the arguments contained by the part elements of the messages are associated with the element definition/declarations or type definitions of the XSD. Since each element or type definition in the schema is assigned to a weight value that reflects their complexity degrees and is associated with the arguments, the $wp_i$ has the same weight value as the associated element or type definitions of XSD. That is

$$wp = we, \text{ if part has element reference to the schema element declaration} \tag{6}$$

$$= wt, \text{ if part has type reference to the schema element definition} \tag{7}$$

$$= 0, \text{ if the message has no <part> element} \tag{8}$$

*we* is the weight value of the associated element declaration [27] and *wt* is the weight value of the associated type definition in the schema embedded in or imported to WSDL. The weight value of a type *wt* hence can be defined as

$$wt = wc, \text{ if the type is complex-type definition} \tag{9}$$

$$= ws, \text{ if the type is simple-type definition} \tag{10}$$

The following example demonstrates how to calculate DW (WSDL) value for a given WSDL document.

```
....
<message name="GetGSInformationRequest">
        <part name="ID" type="xsd:string" />
        <part name="Password" type="xsd:string" />
</message>
<message name="StringResponse">
        <part name="Reponse" type="xsd:string" />
</message>
...
<portType name="GSNPortType">
        <!-- Information Operation -->
        <operation name="GetGSInformation">
                <input message="tns:GetGSInformationRequest" />
                <output message="tns:StringResponse" />
        </operation>
.....
```

**Fig. 3** *Description of the operation GetGSInformation in an example WSDL document*

```
<wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://terraserver-
usa.com/LandmarkServer/">
  <s:element name="GetLandmarkTypes">
      <s:complexType />
  </s:element>
  <s:element name="GetLandmarkTypesResponse">
      <s:complexType>
          <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="GetLandmarkTypesResult"
type="tns:ArrayOfLandmarkType" />
          </s:sequence>
      </s:complexType>
 </s:element>
 <s:complexType name="ArrayOfLandmarkType">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="LandmarkType"
type="tns:LandmarkType" />
      </s:sequence>
</s:complexType>
<s:complexType name="LandmarkType">
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="ShapeType"
type="tns:ShapeType" />
        <s:element minOccurs="0" maxOccurs="1" name="Type" type="s:string" />
      </s:sequence>
</s:complexType>
<s:simpleType name="ShapeType">
   <s:restriction base="s:string">
       <s:enumeration value="Null" />
       <s:enumeration value="Point" />
       <s:enumeration value="PolyLine" />
       <s:enumeration value="Polygon" />
  </s:restriction>
</s:simpleType>
....
</wsdl:types>
<wsdl:message name="GetLandmarkTypesSoapIn">
   <wsdl:part name="parameters" element="tns:GetLandmarkTypes" />
 </wsdl:message>
 <wsdl:message name="GetLandmarkTypesSoapOut">
   <wsdl:part name="parameters" element="tns:GetLandmarkTypesResponse" />
 </wsdl:message>
......
<wsdl:portType name="LandmarkServiceSoap">
  <wsdl:operation name="GetLandmarkTypes">
     <wsdl:input message="tns:GetLandmarkTypesSoapIn" />
     <wsdl:output message="tns:GetLandmarkTypesSoapOut" />
  </wsdl:operation></wsdl:portType>
.....
```

**Fig. 4** *Description of the operation GetLandmarkTypes in an example WSDL document*

*Example 1:* In Figs. 3 and 4 we only show the description of one operation and its request and respond messages of the real-life WSDL documents to demonstrate the calculation of DW value. We also evaluate the APO, OPS, DAR and ARS values for these two WSDLs. The operation GetGSInformation has two arguments namely ID and password for its input message and one argument named response for its output message. The data types of these three arguments are defined by the type attribute of part elements of its messages constructs and have an XML schema-type string. In order to evaluate DW (WSDL) we first calculate the complexities of its each message.

$$\begin{aligned} C(\text{'GetGSInformationRequest'}) &= \Sigma_{j=0}^{1} wp_j \\ &= wp_{\text{ID}} + wp_{\text{password}} \\ &= wt_{\text{ID}} + wt_{\text{password}} \\ &= ws_{\text{string}} + ws_{\text{string}} \\ &= 1 + 1 \\ &= 2 \end{aligned}$$

$$\begin{aligned} C(\text{'StringResponse'}) &= \Sigma_{j=0}^{1} wp_j = wp_{\text{response}} \\ &= wt_{\text{response}} = ws_{\text{string}} = 1 \end{aligned}$$

The weight value for a part element is assigned based on the weight value of the argument it describes and the argument



*IET Softw.*, 2011, Vol. 5, Iss. 3, pp. 320–341
doi: 10.1049/iet-sen.2010.0089

described by part element will have a weight of its associated type of XML schema. The weight values for built-in simple types of XML schema are assigned to 1 [27]. Consequently, the arguments that have built-in simple type have a weight value of 1. Thus, the data complexity value for the input message is found as 2 and for the output message it is found 1. Overall the data complexity of the WSDL in Fig. 3 is evaluated by

$$\begin{aligned}\text{DW (WSDL)} &= \Sigma_{i=1}^{2} C(M_i) \\ &= C(\text{'GetGSInformationRequest'}) \\ &\quad + C(\text{'StringResponse'}) \\ &= 2 + 1 \\ &= 3\end{aligned}$$

In Fig. 4 the operation GetLandmarkTypes is described in the second real-life WSDL document. While the part element in the input message of the operation associates the input argument with the complex-type GetLandmarkTypes element of the XSD, the output message's part element associates the complex-type GetLandmarkTypesResponse element of XSD for defining the output arguments namely ShapeType and Type. The complexity values [27] for each message evaluated as

$$\begin{aligned}&C(\text{'GetLandmarkTypesSoapIn'}) \\ &= \Sigma_{j=0}^{0} wp_j = wp_{\text{parameters}} \\ &= we_{\text{tns:GetLandmarkTypes}} \\ &= wc_{\text{complexType}} = 1 \\ &C(\text{'GetLandmarkTypesSoapOut'}) \\ &= \Sigma_{j=0}^{0} wp_j = wp_{\text{parameters}} \\ &= we_{\text{tns:GetLandmarkTypesResponse}} \\ &= wc_{\text{tns:GetLandmarkTypesResponse}} = 6\end{aligned}$$

Weight value for the element declaration tns: GetLandmarkTypes of XSD given in Fig. 4 is found as 1 and for the element declaration tns: GetLandmarkTypesResponse in XSD that is associated for the output arguments description of the operation is found as 6. Hence, the overall data complexity value for the WSDL document given in Fig. 4 is

$$\begin{aligned}&W(\text{wsdl}) \\ &= \Sigma_{i=1}^{2} C(M_i) = C(\text{'GetLandmarkTypesSoapIn'}) \\ &\quad + C(\text{'GetLandmarkTypesSoapOut'}) \\ &= 1 + 6 = 7\end{aligned}$$

The APO, OPS, DAR and ARS values for these two WSDLs are found as 3, 1, 1 and 1, respectively.

As can be seen in the example 1, even both WSDLs have equal APO, OPS, DAR and ARS values, DW (WSDL) values of them are not equal owing to the complexities of the arguments' data structures. Thus, DW metric can better differentiate the WSDLs in terms of their data complexities. For the empirical validation and to verify the usefulness of DW, we have applied DW on 56 different WSDL files and compared with APO and OPS metrics. The graph shown in Fig. 5 depicts the comparison result between the OPS and DW (WSDL) metrics and Fig. 6 shows DW (WSDL) and APO comparison result for analysed WSDL documents. Note that these graphs are drawn in logarithmic scale with base 10 and WSDL files are ordered according to the values of OPS and APO metrics, respectively.

If we compare the complexity values of OPS and DW (Fig. 5), we can easily observe that several WSDLs have the equal OPSs values but for the same WSDLs, DW values are different. For example, OPS value for WSDL ids 5, 6, 39 and 48 is 2; however, DW values for the same WSDL are different and are 12, 50, 7 and 81, respectively. Further, since the graph in Fig. 5 is ordered with increasing values of operations, DW values are not increasing accordingly. The lowest OPS value, which is 1 belongs to the WSDL ids 19, 21 and 22; however, DW values for those WSDLs are 32, 89 and 8, respectively. Similarly, in Fig. 6, the lowest value of APO belongs to WSDL ids 20, 39 and 43 and is 2. The corresponding DW values for these WSDLs are 20, 7 and 32, respectively.

The graphs in Figs. 5 and 6 reflect that the data complexity of a web service does not increase with increase in the number of arguments or operations. This result verifies our claim that we stated in the beginning.

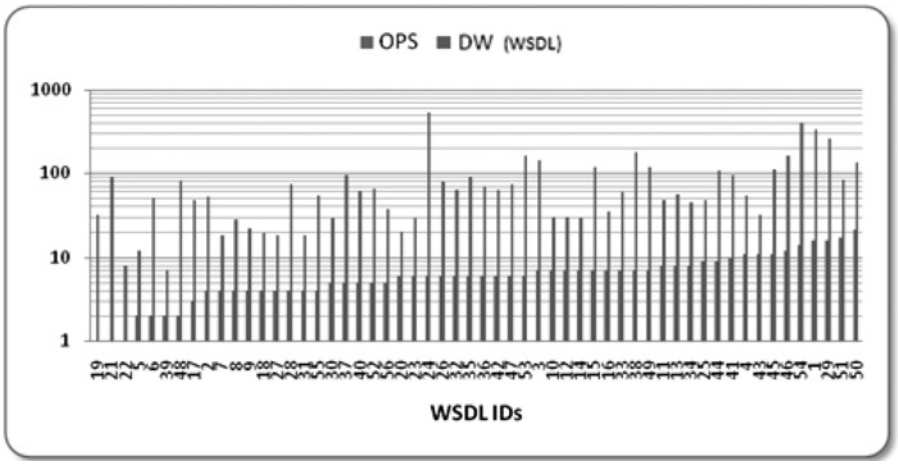


**Fig. 5** *Comparative graph between OPS and DW (DSWL)*

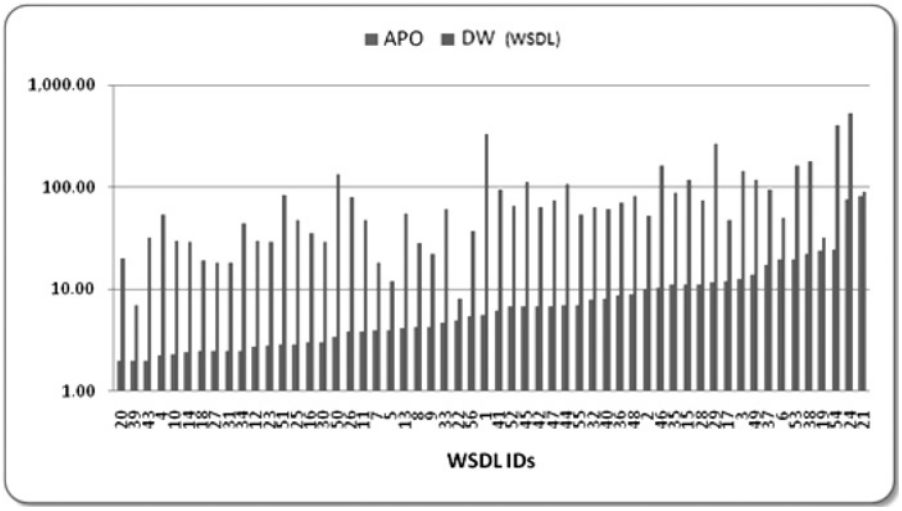


**Fig. 6** *Comparative graph between APO and DW (WSDL)*

### 4.2 DMR metric

Consider the real-life WSDL document shown in Fig. 18 in the Appendix. While inspecting this WSDL, we observed that in the some of the message descriptions the same XSD element declaration is associated with some of the arguments to define these arguments' data structures. In other words, these messages have the same structures since the number of arguments they contain and the data complexity values owing to contained argument's data-type definitions are equal. Intuitively one can expect that as the number of similar-structured messages increases the data complexity of a WSDL decreases since it is easier to trace, understand and remember similar-structured messages than that of various-structured ones as a result of gained familiarity with repetitive messages.

Since our DW (WSDL) metric does not capture the effects of similar-structured messages on the data complexity of a web service, we introduce the following DMR metric that is derived from distinct message count (DMC) metric to capture this aspect. We first provide a definition for DMC metric that considers structures of the messages and is adapted from DAC metric [25]. Note that while DAC metric considers on distinct arguments but ignores the complexities of them, DMC considers distinct messages structures characterised according to the number of arguments they contain and the data complexities of the contained arguments.

*Definition 2:* The DMC metric can be defined as the number of distinct structured messages represented by [$C$ ($M$), arg] pair reflecting the message's complexity value $C$ ($M$) and total number of arguments arg that the message contains.

The messages of the operations described in the WSDL document in Fig. 18 in the Appendix are represented by labelled pairs to be identified based on DMC definition. As an example, the message GetMessagesMessengerHeader can be represented by G3 (4, 3) pair meaning that it has data complexity value of 4 and contains 3 arguments for its related operation. The messages represented by the same pair can be regarded as having equal data complexity. Hence, DMC value for this example WSDL document is found as seven that is equal to the number of pairs and the pairs are labelled as G1, G2, . . . , G7.

*Definition 3:* DMR can reflect the data complexities of a web service as a result of having similar-structured messages

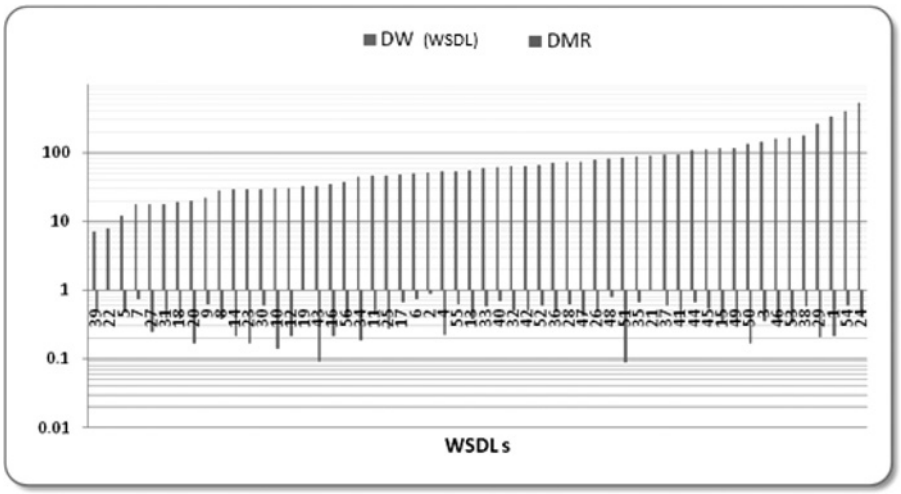


**Fig. 7** *Comparative graph between DW and DMR values for analysed WSDLs*




doi: 10.1049/iet-sen.2010.0089

exchanged by the operations and is defined as

$$\text{DMR} = \frac{\text{DMC}}{n_{\text{m}}} \tag{11}$$

where $n_{\text{m}}$ is the total number of messages in WSDL.

*Example 2:* The DMR metric value of the WSDL document shown in Fig. 18 in Appendix is found as

$$\begin{aligned}\text{DMR} &= 7/21 \\ &= 0.33\end{aligned}$$

For a given web service having lower DMR implies that the service has a lower data complexity since the messages of the operations are consistent in structures it is easier to remember similar-structured messages. Again for empirical validation of DMR we analysed 56 WSDL documents and compared DMR with DW, APO and OPS metrics to verify that DMR metric can be useful to differentiate web services that have equal DW, APO and OPS values in terms of their data complexities. In Fig. 7, the graph drawn according to the collected statistics (Table 3) depicts the comparison result between DMR and DW. DMR metric differentiates the WSDLs based on similarly structured messages in WSDL files and DW metric differentiates WSDLs based on their internal architecture of the messages. This is the reason why several WSDLs that have same DW values their DMR values are different. For example, WSDL ids 7, 27 and 31 have the same DW values (i.e. 18); however, their corresponding DMR values are 0.75, 0.25 and 0.40 (Fig. 7).

The graph in Fig. 8 shows the comparison result between DMR and APO. From this graph we can very easily observe that DMR values are the better representation of complexity of WSDL files, since DMR represents the psychological complexity based on similarly structured messages and therefore provide different values where for the same WSDLs, APO and OPS values are same. If we compare the values of DMR and APO, we find that WSDL ids 20, 39 and 43 have equal APO value (i.e. 2); however, their DMR values are different and are 0.17, 0.50 and 0.09, respectively (Fig. 8). For similar reason, OPSs values of WSDL ids 2, 7, 8, 9, 18, 27, 28, 31, 55 are equal and 4; however, the corresponding DMR values are 0.88, 0.75, 0.38, 0.63, 0.38, 0.25, 0.63, 0.40 and 0.63, respectively.

### 4.3 ME metric

Now, let us assume that we have two WSDL documents each comprising 12 messages in total and equal DMC value that is, both have equal DMR value. While the occurrences of the distinct structured messages are 10, 1 and 1 in the first WSDL, the second WSDL has the occurrences of 4 for each distinct structured message. It is clear that comprehending the first WSDL is easier than that of the second one owing to the fact that repetition of the same-structured messages makes the developer more familiar with the architecture of the WSDL and results in ease of understandability. In such a case neither DW nor DMR metrics can be sufficient to capture the data complexity of WSDL owing to the repetition of similar-structured messages.

The ME metric is intended to measure the complexity caused by occurrences of similar-structured messages. We adapted entropy [28–29] to measure diversity in message structures described in a WSDL; hence, the lower ME value shows that the messages are consistent in structure which means that data complexity of a web service is lower than that of the others having equal APO, OPS or DMR values. Compared with DMR metric, the ME provides better differentiation between web services in terms of their data complexity.

*Definition 4:* ME of a given WSDL is

$$\text{ME (wsdl)} = -\sum_{i=1}^{\text{DMC}} P(dm_i)\log_2 P(dm_i) \tag{12}$$

$$\text{P}(dm_i) = \frac{\text{nom}_i}{n_{\text{m}}} \tag{13}$$

where, P $(dm_i)$ is the probability of $i$th distinct message structure represented by [$C$ ($M$), arg] pair in WSDL. The probability of a message represented by [$C$ ($M$), arg] is equal to the number of occurrences of its associated pair,

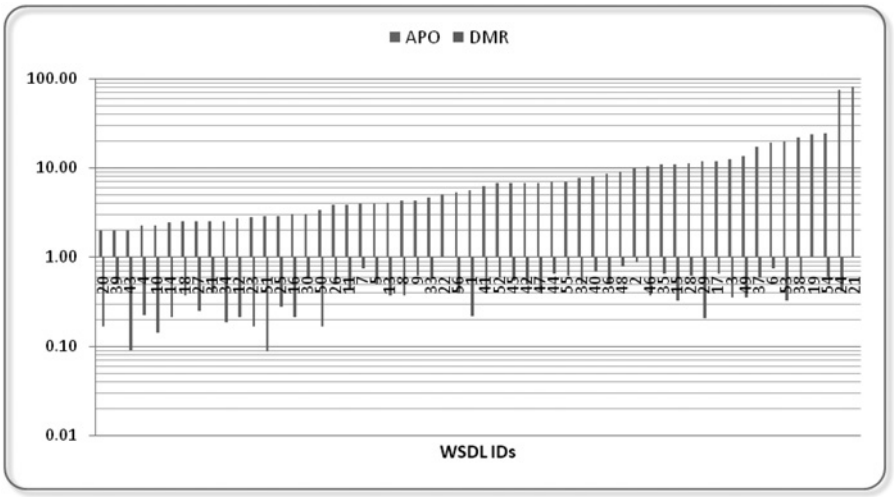


**Fig. 8** *Comparative graph between DMR and APO metrics values for analysed WSDLs*

$\text{nom}_i$ divided by the total number of messages $n_m$ in a WSDL. The possible values for ME will be in the range $\log_2(n_m) \leq \text{ME} \leq 0$. The lower the ME value the WSDL has, more consistent is the message structure and the less data flow complexity in WSDL's interface.

*Example 3:* The WSDL file shown in Fig. 18 in the Appendix has seven distinct messages structures represented by the set of pairs G1, G2, ... , G7, that is DMC = 7. The messages that are represented by the same pairs are grouped into the same class that has the same name with the pair. For example, the messages represented by the G3 pair are grouped into the class named G3. Since the number of messages represented by G3 pair is 8, the member's count of this class is 8 that gives the occurrences of the pair G3. The classes that consist of the same pairs and number of their members are listed below

$$\begin{aligned}
\text{G1} &= 3[C(M) = 1, \arg = 1] \\
\text{G2} &= 4[C(M) = 14, \arg = 9] \\
\text{G3} &= 8[C(M) = 4, \arg = 3] \\
\text{G4} &= 2[C(M) = 2, \arg = 1] \\
\text{G5} &= 1[C(M) = 13, \arg = 9] \\
\text{G6} &= 2[C(M) = 3, \arg = 1] \\
\text{G7} &= 1[C(M) = 3, \arg = 2]
\end{aligned}$$

and ME value for the WSDL file is

$$\begin{aligned}
\text{ME}(\text{'u7.wsdl'}) &= -\sum_{i=1}^{7} P(dm_i) \log_2 P(dm_i) \\
&= -[(3/21)^* \log_2(3/21) + (4/21)^* \log_2(4/21) \\
&\quad + (8/21)^* \log_2(8/21) + (2/21)^* \log_2(2/21) \\
&\quad + (1/21)^* \log_2(1/21) + (2/21)^* \log_2(2/21) \\
&\quad + (1/21)^* \log_2(1/21)] \\
&= 2.45161
\end{aligned}$$

In Fig. 9, the graph depicts the comparison result between DMR and ME metrics that is evaluated according to the collected statistics (Table 3) of analysed 56 WSDL documents. From the graph in Fig. 9, it can be observed that ME is proportional to DMR in general, that is, as DMR value increases so does ME value. However, the ME metric can differentiate WSDLs in terms of their data complexities when the DMR values of them are equal. For example, DMR values for WSDL ids 26, 32, 36 and 42 are equal and 0.50. On the other hand, ME values for these WSDLs are 2.36, 2.52, 2.42 and 2.46, respectively. We have also demonstrated the comparison of ME metric with APO and OPS metrics in Figs. 10 and 11. From the graphs depicted in Figs. 10 and 11, we can see that complexities of WSDLs that have equal OPS and APO metrics values are different evaluated by ME metric. From Fig. 10 we can easily observe that, WSDL ids 10, 12, 14, 15, 16, 33, 38 and 49 all have the same OPS values (i.e. 7) but different ME values, that is, 0.59, 1.53, 1.29, 2.45, 1.29, 2.75, 2.81, 2.81 and 2.22, respectively. Similarly in Fig. 11, WSDL ids 20, 39 and 43 have equal APO values, that is, 2.0 but different ME values, that is, 0.92, 0.81 and 0.99. In the graphs (Figs. 10 and 11), we can find several such types of examples. All these examples prove that ME metric differentiate WSDLs in better way in comparison with APO and OPS metrics.

### 4.4 MRS metric

MRS metric is intended to analyse variety in structures of WSDL documents. It is adapted from ARS metric that is proposed by Boxall and Araban [26] as an interface metric for reusability analysis of software components and used to measure the consistency of the arguments that software interfaces use. In his work Boxall identified each argument by (name, type) pairs and determined the unique pairs. By considering frequencies of these unique pairs he established ARS metric to measure the consistency of naming and typing of arguments. He concluded that while higher ARS indicates more consistency in argument declaration lower ARS indicates more specialised functionality of interfaces. For example, consider two interfaces with three unique arguments and twelve arguments in total. The first interface has three distinct arguments that are used four times. On the other hand, the second interface has one distinct argument that is used ten times, and two other distinct arguments are used only once. It may be argued that the second interface is more consistent in argument declarations. With this example, he drew a conclusion that

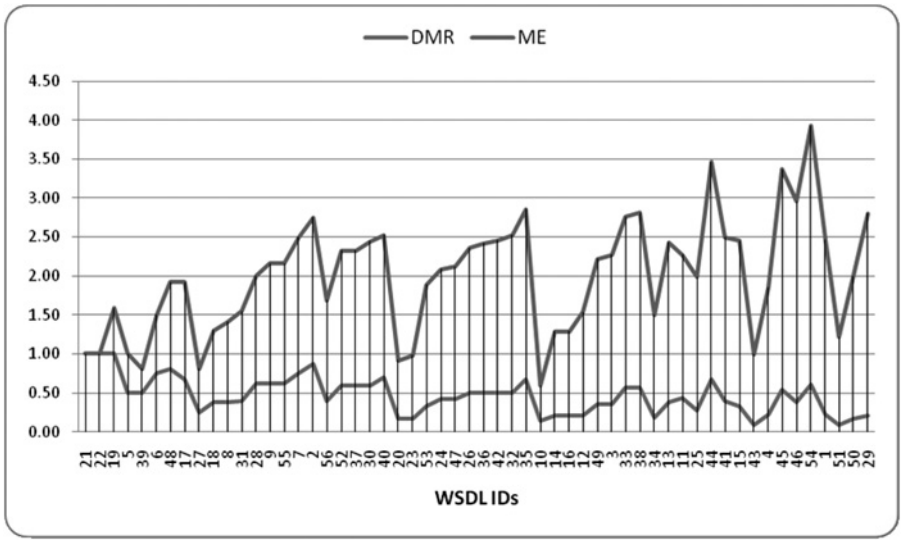


**Fig. 9** *Comparative graph between DMR and ME metric values for analysed WSDLs*




doi: 10.1049/iet-sen.2010.0089

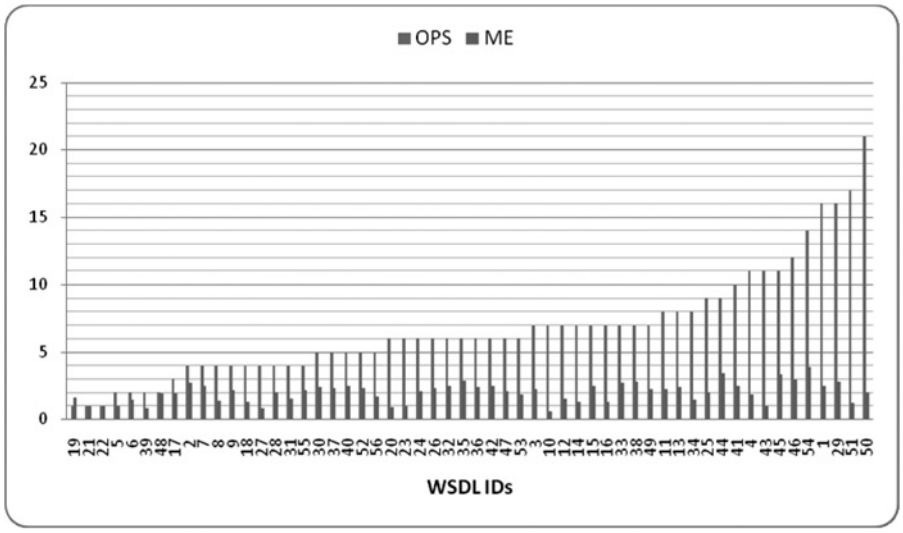


**Fig. 10** *Comparative graph between ME and OPS metric values for analysed WSDLs*

more consistent argument declarations allow better knowledge transferability and make it easier to understand the interfaces.

Following the similar approach, (MRS) can be used for measuring the data complexity of WSDL documents and can be defined as

$$\text{MRS} = \frac{\sum_{i=1}^{\text{DMC}} \text{nom}_i^2}{n_{\text{m}}} \tag{14}$$

where $\text{nom}_i$ is the occurrence of the $i$th distinct message structure represented by [$C$ ($M$), arg] pair and $n_{\text{m}}$ is the total number of messages in the WSDL document. The possible values for MRS will be in the range $1 \leq \text{MRS} \leq n_{\text{m}}$. The relation between MRS and ME is that while higher MRS value indicates lower data complexity of WSDL, the higher ME metric value indicates the opposite. In other words, higher MRS and lower ME show that the developer makes less effort to understand the messages structures owing to the repetition of similar messages.

*Example 4:* For the WSDL file shown in Fig. 18 in the Appendix. The MRS value is calculated by considering the messages listed in Example 3

$$\text{MRS} = \frac{\sum_{i=1}^{7} \text{nom}_i^2}{21} = (3^2 + 4^2 + 8^2 + 2^2 + 1^2 + 2^2 + 1^2)/21$$
$$= 83/21 = 3.95238$$

We evaluate the graph depicted in Figs. 12 and 13 to show the relation between MRS-ME and MRS-DMR, after analysing 56 WSDL documents. The analysis of graph (Fig. 12) shows that MRS metric differentiates those services, whose ME values are equal, for example for id, 28 and 50, the MRS values are 2.5 and 13.0; however, their ME values are equal and 2.0. In general, graphs (Figs. 12 and 13) show the inverse relation between MRS–ME and MRS–DMR metrics. This relation of MRS with ME and DMR stems from the fact that all these metrics are based on the similarity of structures of the message.

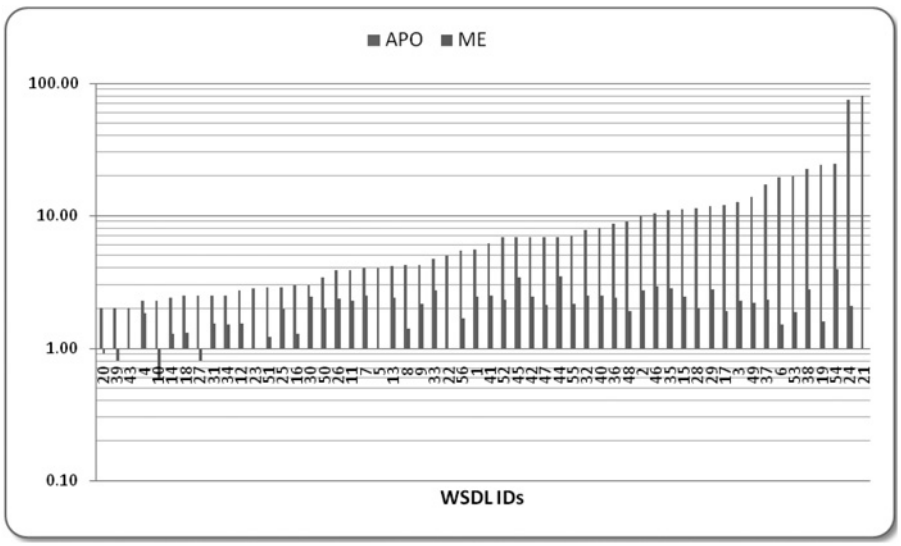


**Fig. 11** *Comparative graph between ME and APO metric values for analysed WSDLs*

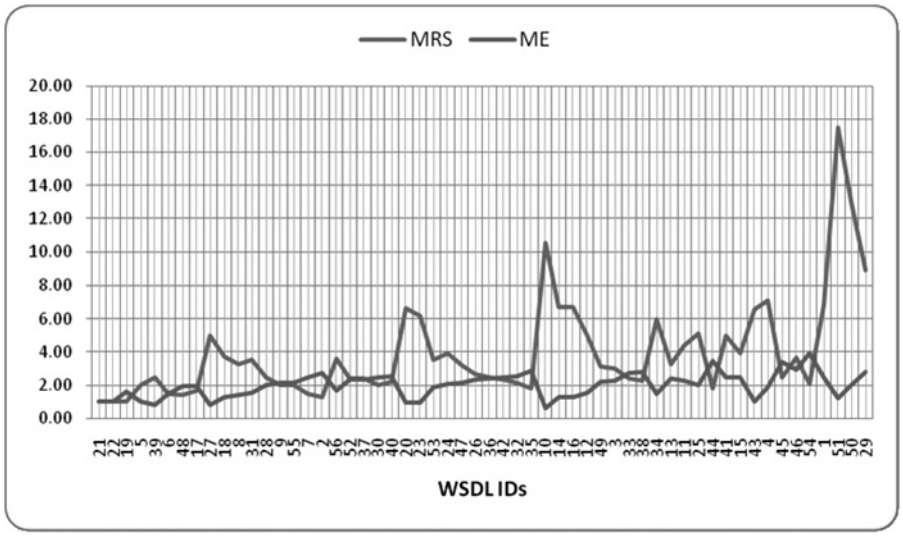


**Fig. 12** *Comparative graph between MRS and ME metric values for analysed WSDLs*

## 5 Theoretical validations of proposed metrics

In this section we evaluate and validate our proposed metrics theoretically. For theoretical validation, several researchers proposed different criteria [30–40], to which the proposed software metric should adhere. However, in general all the aforementioned criteria suggest that the metric should fulfil some basic requirements based on the measurement theory perspective. In the next section, we evaluate our metrics against Weyuker's properties [33].

In the evaluation process, we use three example WSDL documents given Figs. 14–16, respectively.

The WSDL document given in Fig. 16 is designed by concatenating the WSDLs given in Figs. 14 and 15. In order to classify distinct messages, each of them is represented by [*C* (*M*), arg] pair and labelled as G1, G2 and G3 according to their representations. Having labelled these messages DMC values for these three WSDL are found as 2, 2 and 3, respectively.

### 5.1 Evaluation of proposed metrics by Weyuker's properties

Weyuker [33] suggested nine properties to provide a formal approach to determine the effectiveness of various software complexity measures. A good complexity measure should satisfy most of Weyuker's properties. Although Weyuker properties are criticised by several researchers [40–42], these properties are still undergoing research [43–46] and are used by several researchers [47–51]. We also used these properties to evaluate our proposed measure.

*Property 1:* (∃P) (∃Q) (|P| ≠ |Q|). Where P and Q are program bodies. As can be observed from Table 3 it is obvious that for different Schema documents different DW, DMR, ME and MRS values are evaluated, and hence this property is hold by all of our metrics.

*Property 2:* Let *c* be a non-negative number then there are only finitely many programs of complexity *c*. All WSDL documents consist of only finite number of messages and arguments taken by these messages and our metrics depend on the structures of messages and their arguments. Thus, given that a WSDL contains only finitely many messages and arguments, there are only finitely many WSDLs being equal to the measure *c*. Hence, our metrics hold this property.

*Property 3:* There are distinct programs P and Q such that |P| = |Q|.

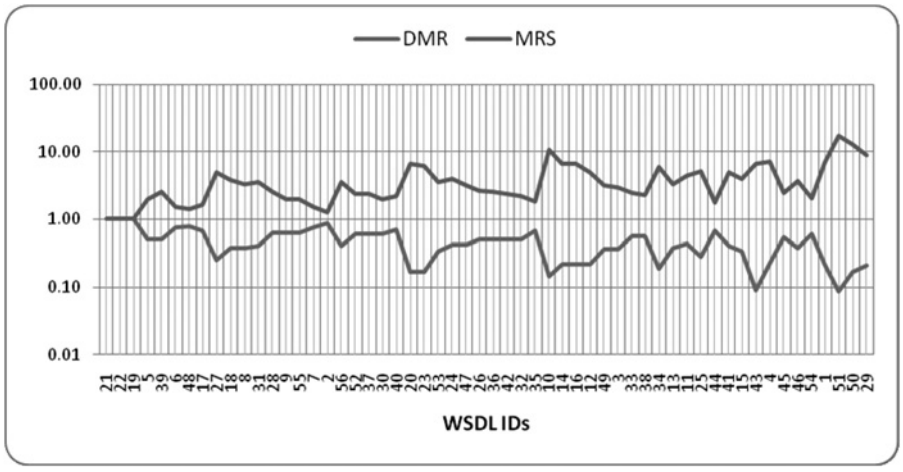


**Fig. 13** *Comparative graph between DMR and MRS metric values for analysed WSDLs*




doi: 10.1049/iet-sen.2010.0089

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:ex1"
        xmlns:tns="urn:ex1"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
<wsdl:types>
 <xsd:schema targetNamespace="urn:ex1"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:complexType name="Address">
    <xsd:sequence>
     <xsd:element name="streetNum" type="xsd:int"/>
     <xsd:element name="streetName" type="xsd:string"/>
     <xsd:element name="city" type="xsd:string"/>
     <xsd:element name="state" type="xsd:string"/>
    </xsd:sequence>
   </xsd:complexType>
 </xsd:schema>
</wsdl:types>
<wsdl:message name="GetAddressRequest">
 <wsdl:part name="name" type="xsd:string"/><!—G1(1,1)- ->
</wsdl:message>
<wsdl:message name="GetAddressResponse">
 <wsdl:part name="address" type="tns:Address"/><!—G2(5,4)- ->
</wsdl:message>
<wsdl:portType name="AddressBook">
 <wsdl:operation name="getAddress">
  <wsdl:input message="tns:GetAddressRequest"/>
  <wsdl:output message="tns:GetAddressResponse"/>
 </wsdl:operation>
</wsdl:portType>
</wsdl:definitions>
```

**Fig. 14** *Example WSDL document used in theoretical validation (1)*

As can be seen from Table 3, different WSDL documents may have same DW, DMR, ME and MRS values and hence this property is also satisfied.

*Property 4:* $(\exists P)\ (\exists Q)\ (P \equiv Q\ \&\ |P| \neq |Q|)$.

Based on this property even two web services have the same functionality their complexities measured by DW, DMR, ME and MRS metrics may be different. The main consideration of these metrics is complexity owing to the message structures and that complexity based on the complexity of data representation of the arguments. The arguments that WSDL's messages take can be defined in the schema with different implementation and hence may have different complexity levels. Therefore our metrics evaluate different complexity values for the two WSDL having same functionalities. Thus, this property is also satisfied by all of four metrics.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:ex2"
        xmlns:tns="urn:ex2"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
<wsdl:types>
      <xsd:schema targetNamespace="urn:ex2"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:complexType name="Phone">
    <xsd:sequence>
     <xsd:element name="areaCode" type="tns:int"/>
     <xsd:element name="exchange" type="xsd:int"/>
     <xsd:element name="number" type="xsd:int"/>
    </xsd:sequence>
   </xsd:complexType>
 </xsd:schema>
</wsdl:types>
<wsdl:message name="GetPhoneRequest">
 <wsdl:part name="name" type="xsd:string"/><!—G1(1,1)- ->
</wsdl:message>
<wsdl:message name="GetPhoneResponse">
 <wsdl:part name="phone" type="tns:Phone" /><!—G2(4,3)- ->
</wsdl:message>
<wsdl:portType name="getPhone">
   <wsdl:operation name="getPhone">
   <wsdl:input message="tns:GetPhoneRequest"/>
   <wsdl:output message="tns:GetPhoneResponse"/>
 </wsdl:operation>
</wsdl:portType>
</wsdl:definitions>
```

**Fig. 15** *Example WSDL document used in theoretical validation (2)*

```
<wsdl:definitions targetNamespace="urn:ex1ex2"
        xmlns:tns="urn:ex1ex2"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
<wsdl:types>
 <xsd:schema targetNamespace="urn:ex1ex2"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:complexType name="Address">
    <xsd:sequence>
     <xsd:element name="streetNum" type="xsd:int"/>
     <xsd:element name="streetName" type="xsd:string"/>
     <xsd:element name="city" type="xsd:string"/>
     <xsd:element name="state" type="xsd:string"/>
    </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Phone">
    <xsd:sequence>
     <xsd:element name="areaCode" type="tns:int"/>
     <xsd:element name="exchange" type="xsd:int"/>
     <xsd:element name="number" type="xsd:int"/>
    </xsd:sequence>
   </xsd:complexType>
 </xsd:schema>
</wsdl:types>
<wsdl:message name="GetAddressRequest">
   <wsdl:part name="name" type="xsd:string"/><!—G2(1,1)- ->
 </wsdl:message>
<wsdl:message name="GetAddressResponse">
   <wsdl:part name="address" type="tns:Address"/><!—G3(5,4)- ->
</wsdl:message>
<wsdl:message name="GetPhoneRequest">
   <wsdl:part name="name" type="xsd:string"/><!—G2(1,1)- ->
</wsdl:message>
<wsdl:message name="GetPhoneResponse">
   <wsdl:part name="phone" type="tns:Phone" /><!—G1(4,3)- ->
</wsdl:message>
<wsdl:portType name="AddressBook">
   <wsdl:operation name="getAddress">
    <wsdl:input message="tns:GetAddressRequest"/>
    <wsdl:output message="tns:GetAddressResponse"/>
   </wsdl:operation>
   <wsdl:operation name="getPhone">
    <wsdl:input message="tns:GetPhoneRequest"/>
    <wsdl:output message="tns:GetPhoneResponse"/>
   </wsdl:operation>...
```

**Fig. 16** *Combination of WSDLs given in Figs. 14 and 15*

*Property 5:* $(\forall P)\ (\forall Q)\ (|P| \leq |P;\ Q|$ and $|Q| \leq |P;\ Q|)$.

This property states that the complexity values of two WSDL documents P, Q should be less than or equal to the complexity of the composition of the two WSDL documents. Consider the WSDL documents given in Figs. 14–16 representing P, Q and P; Q. Note that the WSDL document given in Fig. 16 is the combination of the WSDL given in

**Table 1** Metrics values of WSDL documents given in Figs. 14–16

| WSDL | #of messages | DMC | DW | DMR | ME | MRS |
|---|---|---|---|---|---|---|
| Fig. 14 | 2 | 2 | 6 | 1 | 1 | 1 |
| Fig. 15 | 2 | 2 | 5 | 1 | 1 | 1 |
| Fig. 16 | 4 | 3 | 11 | 0.75 | 1.5 | 1.5 |

Figs. 14 and 15. The metrics values given in Table 1 for these three WSDL documents are calculated as

$$DW(\text{'Fig. 14.wsdl'}) = \sum_{i=1}^{2} C(M_i)$$
$$= C(\text{'GetAddressRequest'}) + C(\text{'GetAddressResponse'})$$
$$= wp_{\text{name}} + wp_{\text{address}} = ws_{\text{string}} + wc_{\text{Adress}} = 1 + 5 = 6$$

$$DW(\text{'Fig. 15.wsdl'}) = \sum_{i=1}^{7} C(M_i)$$
$$= C(\text{'GetPhoneRequest'}) + C(\text{'GetPhoneResponsee'})$$
$$= wp_{\text{name}} + wp_{\text{phone}} = ws_{\text{string}} + wc_{\text{Phone}} = 1 + 4 = 5$$

$$DW(\text{'Fig. 16.wsdl'}) = \sum_{i=1}^{7} C(M_i)$$
$$= C(\text{'GetAddressRequest'}) + C(\text{'GetAddressResponse'}) + C(\text{'GetPhoneRequest'}) + C(\text{'GetPhoneResponsee'})$$
$$= wp_{\text{name}} + wp_{\text{address}} + wp_{\text{name}} + wp_{\text{phone}}$$
$$= ws_{\text{string}} + wc_{\text{Adress}} + ws_{\text{string}} + wc_{\text{Phone}}$$
$$= 1 + 5 + 1 + 4 = 11$$

$$\text{DMR}_{\text{Fig.14.wsdl}} = \text{DMR}_{\text{Fig.15.wsdl}} = \frac{\text{DMC}}{n_{\text{m}}} = 2/2 = 1$$

$$\text{DMR}_{\text{Fig.15.wsdl}} = 3/4 = 0.75$$

$$ME(\text{'Fig. 14.wsdl'}) = ME(\text{'Fig. 15.wsdl'}) = -[(1/2)^* \log_2(1/2) + (1/2)^* \log_2(1/2)] = 1$$

$$ME(\text{'Fig. 16.wsdl'}) = -[(1/4)^* \log_2(1/4) + (2/4)^* \log_2(2/4) + (1/4)^* \log_2(1/4)] = 1.5$$

$$\text{MRS}(\text{'Fig. 14.wsdl'}) = \text{MRS}(\text{'Fig. 15.wsdl'}) = (1^2 + 1^2)/2 = 1$$

$$\text{MRS}(\text{'Fig. 16.wsdl'}) = (1^2 + 2^2 + 1^2)/4 = 1.5$$

As can be observed from Table 1, except for DMR metric all other three metrics satisfy this property.

*Property 6:* (∃P) (∃Q) (∃R) (|P| = |Q|) & (|P; R| ≠ |Q; R|).

This property asserts that we can find two WSDL documents of equal DW, DMR, ME and MRS values which when separately concatenated to a same third WSDL document yields the WSDL of different DW, DMR, ME and MRS values. Assume we have three WSDL documents; P, Q and R that do not have elements in common. The first WSDL document P.wsdl having four distinct message structures and its messages are labelled as G1, $G_2$, $G_3$ and G4. The number of occurrences of these messages are 3, 3, 1 and 1, respectively. Note that messages are represented by [*C* (*M*), arg] pair where *C* (*M*) is the complexity of a message owing to the complexity of its part element's data structure defined in the schema. The second WSDL, Q.wsdl, has messages completely different in structure from the messages of P.wsdl and its messages' with the number of occurrences are $G_5 = 3$, $G_6 = 3$, $G_7 = 1$ and $G_8 = 1$. The last WSDL, R.wsdl has similar message structures as P.wsdl has and its messages are labelled as: $G_1 = 3$, $G_2 = 3$, $G_3 = 1$ and $G_4 = 1$. Note that Q and R do not have any message structures in common. We can find that the DW, DMR, ME and MRS values for these tree WSDLs are the same.

The DW, ME, DMR and MRS metrics for the resulting WSDL that is a combination of P, R and Q; R will be

$$\text{DW}_{\text{P;R}} = \text{DW}_{\text{Q;R}}$$

Hence, DW metric does not satisfy this property.

Since P and R have similar message structures the combined WSDL has four different message structures ($G_1 = 6$, $G_2 = 6$, $G_3 = 2$, $G_4 = 2$) in total, so

$$\text{ME}_{\text{P;R}} = -\sum_{i=1}^{4} P(dm_i) \log_2 P(dm_i)$$
$$= -[(6/16)^* \log_2(6/16) + (6/16)^* \log_2(6/16) + (2/16)^* \log_2(2/16) + (2/16)^* \log_2(2/16)]$$
$$= 1.81128$$

The WSDLs Q and R have different message structures and combined WSDL has eight message structures, namely $G_1 = 3$, $G_2 = 3$, $G_3 = 1$, $G_4 = 1$, $G_5 = 3$, $G_6 = 3$, $G_7 = 1$ and $G_8 = 1$. So

$$\text{ME}_{\text{Q;R}} = -\sum_{i=1}^{8} P(dm_i) \log_2 P(dm_i)$$
$$= -[(3/16)^* \log_2(3/16) + (3/16)^* \log_2(3/16) + (1/16)^* \log_2(1/16) + (1/16)^* \log_2(1/16) + (3/16)^* \log_2(3/16) + (3/16)^* \log_2(3/16) + (1/16)^* \log_2(1/16) + (1/16)^* \log_2(1/16)]$$
$$= 2.81128$$

$\text{ME}_{\text{P;R}} \neq \text{ME}_{\text{Q;R}}$ and hence ME metric holds this property.

$$\text{DMR}_{\text{P;R}} = 4/16 = 0.25$$
$$\text{DMR}_{\text{Q;R}} = 8/16 = 0.5$$

$\text{DMR}_{\text{P;R}} \neq \text{DMR}_{\text{Q;R}}$ and hence DMR metric does adhere to




doi: 10.1049/iet-sen.2010.0089

this property.

$$\mathrm{MRS}_{\mathrm{P;R}} = \frac{\Sigma_{i=1}^{4}\mathrm{nom}_i^2}{16} = (6^2 + 6^2 + 2^2 + 2^2)/16 = 80/16 = 5$$

$$\mathrm{MRS}_{\mathrm{P;R}} = \frac{\Sigma_{i=2}^{8}\mathrm{nom}_i^2}{16} = (3^2 + 3^2 + 1^2 + 1^2 + 3^2 + 3^2 + 1^2 + 1^2)/16 = 40/16 = 2.5$$

$\mathrm{MRS}_{\mathrm{P;R}} \neq \mathrm{MRS}_{\mathrm{Q;R}}$ and hence MRS metric does also adhere to this property.

*Property 7:* There are WSDL documents P and Q such that Q is formed by permuting the order of the statement of P and ($|P| \neq |Q|$).

Consider the WSDL document given in Fig. 16. If we modify the argument type that the message named GetPhoneRequest takes and its type is derived by restriction without applying any facet from built-in simple type string of W3C XML schema then we evaluate a new distinct structured message and its label is changed from G2 to G4. The modified part of this WSDL version is shown in Fig. 17. In this case the weight value of this message reflecting its complexity degree becomes 2; the DMC value is found as 4 and the DW, DMR, ME and MRS metric values are

$$DW(\text{'Fig. 17'}) = 4 + 1 + 5 + 2 = 12$$

$$\mathrm{DMR}(\text{'Fig. 17'}) = 4/4 = 1$$

$$\mathrm{ME}(\text{'Fig. 17'}) = -[(1/4)^* \log_2(1/4) + (1/4)^* \log_2(1/4) + (1/4)^* \log_2(1/4) + (1/4)^* \log_2(1/4)] = 2$$

$$\mathrm{MRS}(\text{'Fig. 17'}) = (1^2 + 1^2 + 1^2 + 1^2)/4 = 1$$

What we have found for DW, DMR, ME and MRS metrics values for the WSDL document given Fig. 16 is 11, 0.75, 1.5 and 1.5, respectively. By modifying this WSDL without affecting its functionality and the value space of its arguments we evaluated different values for these metrics (12, 1, 2 and 1, respectively). Hence, this property is satisfied by all of our metrics

*Property 8:* Renaming of WSDL cannot change the value of our metrics. As a consequence, this property is also satisfied.

*Property 9:* (∃P) (∃Q) ($|P| + |Q| < |P; Q|$).

From Table 1 one can easily see that our measures do not satisfy the original Weyuker's properties. However, modified version of this property: (∃P) (∃Q) ($|P| + |Q| <= |P; Q|$) is more valuable in evaluating the complexity metrics [37], [39]. Therefore if we refer the same example used in Property 5 and the metrics values for these WSDL documents given in Table 1 it can be observed that only DW metric satisfies this property. (∃P) (∃Q) ($|P| + |Q| < |P; Q|$).

```
...
<xsd:simpleType name="mystring">
      <xsd:restriction base="xsd:string"/>
   </xsd:simpleType>
...
<wsdl:message name="GetPhoneRequest">
   <wsdl:part name="name" type="tns:mystring"/><!—G4(2,1)-->
</wsdl:message>
...
```

**Fig. 17** *Modified part of example WSDL given in Fig. 16*

In this section we have validated our measures through Weyuker's properties. The Table 2 shows a summary of evaluation process through Weyuker's properties for our measures. Note that in Table 2, satisfied Weyuker's properties are marked.

### 5.2 Evaluation through a framework

In the previous section we proved that all proposed metrics are satisfied by eight Weyuker's properties (only DMR satisfy seven properties). However, satisfying Weyuker's properties is necessary, but not a sufficient condition for a good complexity measure [41]. Among several theoretical validation criteria, the framework given by Kaner [30] is adopted for evaluation of our metrics. This approach to metric validation is more practical than the formal approach. The framework is based on answering the following points:

*Purpose of the measures:* The purpose of the measures is to evaluate the quality of web services.

*Scope of usage of the measure:* The proposed DW (WSDL), DMR, ME and MRS metrics are good predictor of understandability of web services, and therefore a valuable contribution for maintainability of web services. Its scope of use is the software development group working specially for web services.

*Identified attribute to measure:* The identified attributes to measure from our suite of metrics are understandability, reliability and maintainability. All these attributes are directly related to the quality of web services.

*Natural scale of the attribute:* The natural scales of the attributes cannot be defined, since it is subjective and requires the development of a common view about them.

*Natural variability of the attribute:* Natural variability of the attributes can also be not defined because of its subjective nature. It is possible that one can develop a sound approach to handle such attribute, but it may not be complete because other factors also exist that can affect the attribute's variability. In this respect, it is difficult to attain knowledge about variability of the attribute.

*Definition of metric:* The metrics have been formally defined in Section 4.

*Measuring instrument to perform the measurement:* We have counted all the parameters of the metrics manually and computed the proposed metrics. Further, we aim at developing a tool/software for measuring the proposed suite of metrics.

**Table 2** Summary of evaluation process for DW, DMR, ME and MRS metrics through Weyuker's properties

| | DW (WSDL) | DMR | ME | MRRS |
|---|---|---|---|---|
| Property 1 | ✓ | ✓ | ✓ | ✓ |
| Property 2 | ✓ | ✓ | ✓ | ✓ |
| Property 3 | ✓ | ✓ | ✓ | ✓ |
| Property 4 | ✓ | ✓ | ✓ | ✓ |
| Property 5 | ✓ | x | ✓ | ✓ |
| Property 6 | x | ✓ | ✓ | ✓ |
| Property 7 | ✓ | ✓ | ✓ | ✓ |
| Property 8 | ✓ | ✓ | ✓ | ✓ |
| Property 9 | ✓ | x | x | x |

**Table 3** Values of the metrics WSDL documents

| ID | REF | #M | #A | OPS | APO | APM | DW (WSDL) | DMC | DMR | MRS | ME (wsdl) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 56 | 32 | 89 | 16 | 5.56 | 2.78 | 334 | 7.0 | 0.22 | 6.81 | 2.47 |
| 2 | 37 | 8 | 40 | 4 | 10.00 | 5.00 | 52 | 7.0 | 0.88 | 1.25 | 2.75 |
| 3 | 57 | 14 | 88 | 7 | 12.57 | 6.29 | 144 | 5.0 | 0.36 | 3.00 | 2.27 |
| 4 | 52 | 22 | 25 | 11 | 2.27 | 1.14 | 54 | 5.0 | 0.23 | 7.09 | 1.86 |
| 7 | 17 | 8 | 16 | 4 | 4.00 | 2.00 | 18 | 6.0 | 0.75 | 1.50 | 2.50 |
| 8 | 8 | 8 | 17 | 4 | 4.25 | 2.13 | 28 | 3.0 | 0.38 | 3.25 | 1.41 |
| 9 | 9 | 8 | 17 | 4 | 4.25 | 2.13 | 22 | 5.0 | 0.63 | 2.00 | 2.16 |
| 5 | 4 | 4 | 8 | 2 | 4.00 | 2.00 | 12 | 2.0 | 0.50 | 2.00 | 1.00 |
| 6 | 10 | 4 | 39 | 2 | 19.50 | 9.75 | 50 | 3.0 | 0.75 | 1.50 | 1.50 |
| 10 | 25 | 14 | 16 | 7 | 2.29 | 1.14 | 30 | 2.0 | 0.14 | 10.57 | 0.59 |
| 11 | 34 | 16 | 31 | 8 | 3.88 | 1.94 | 47 | 7.0 | 0.44 | 4.38 | 2.27 |
| 12 | 26 | 14 | 19 | 7 | 2.71 | 1.36 | 30 | 3.0 | 0.21 | 5.00 | 1.53 |
| 13 | 46 | 16 | 33 | 8 | 4.13 | 2.06 | 55 | 6.0 | 0.38 | 3.25 | 2.43 |
| 14 | 27 | 14 | 17 | 7 | 2.43 | 1.21 | 29 | 3.0 | 0.21 | 6.71 | 1.29 |
| 15 | 44 | 21 | 78 | 7 | 11.14 | 3.71 | 117 | 7.0 | 0.33 | 3.95 | 2.45 |
| 16 | 49 | 14 | 21 | 7 | 3.00 | 1.50 | 35 | 3.0 | 0.21 | 6.71 | 1.29 |
| 17 | 15 | 6 | 36 | 3 | 12.00 | 6.00 | 48 | 4.0 | 0.67 | 1.67 | 1.92 |
| 18 | 14 | 8 | 10 | 4 | 2.50 | 1.25 | 19 | 3.0 | 0.38 | 3.75 | 1.30 |
| 19 | 6 | 3 | 24 | 1 | 24.00 | 8.00 | 32 | 3.0 | 1.00 | 1.00 | 1.58 |
| 20 | 11 | 12 | 12 | 6 | 2.00 | 1.00 | 20 | 2.0 | 0.17 | 6.67 | 0.92 |
| 21 | 12 | 2 | 81 | 1 | 81.00 | 40.50 | 89 | 2.0 | 1.00 | 1.00 | 1.00 |
| 22 | 1 | 2 | 5 | 1 | 5.00 | 2.50 | 8 | 2.0 | 1.00 | 1.00 | 1.00 |
| 23 | 20 | 12 | 17 | 6 | 2.83 | 1.42 | 29 | 2.0 | 0.17 | 6.17 | 0.98 |
| 24 | 56 | 12 | 450 | 6 | 75.00 | 37.50 | 532 | 5.0 | 0.42 | 3.92 | 2.08 |
| 25 | 54 | 18 | 26 | 9 | 2.89 | 1.44 | 47 | 5.0 | 0.28 | 5.11 | 1.99 |
| 26 | 28 | 12 | 23 | 6 | 3.83 | 1.92 | 79 | 6.0 | 0.50 | 2.67 | 2.36 |
| 27 | 13 | 8 | 10 | 4 | 2.50 | 1.25 | 18 | 2.0 | 0.25 | 5.00 | 0.81 |
| 28 | 18 | 8 | 45 | 4 | 11.25 | 5.63 | 73 | 5.0 | 0.63 | 2.50 | 2.00 |
| 29 | 65 | 48 | 189 | 16 | 11.81 | 3.94 | 262 | 10.0 | 0.21 | 8.92 | 2.80 |
| 30 | 29 | 10 | 15 | 5 | 3.00 | 1.50 | 29 | 6.0 | 0.60 | 2.00 | 2.45 |
| 31 | 19 | 8 | 10 | 4 | 2.50 | 1.25 | 18 | 4.0 | 0.40 | 3.50 | 1.55 |
| 32 | 7 | 12 | 47 | 6 | 7.83 | 3.92 | 64 | 6.0 | 0.50 | 2.17 | 2.52 |
| 33 | 50 | 14 | 33 | 7 | 4.71 | 2.36 | 60 | 8.0 | 0.57 | 2.43 | 2.75 |
| 34 | 35 | 16 | 20 | 8 | 2.50 | 1.25 | 44 | 3.0 | 0.19 | 6.00 | 1.50 |
| 35 | 38 | 12 | 66 | 6 | 11.00 | 5.50 | 88 | 8.0 | 0.67 | 1.83 | 2.86 |
| 36 | 39 | 12 | 52 | 6 | 8.67 | 4.33 | 70 | 6.0 | 0.50 | 2.50 | 2.42 |
| 37 | 40 | 10 | 86 | 5 | 17.20 | 8.60 | 94 | 6.0 | 0.60 | 2.40 | 2.32 |
| 38 | 41 | 14 | 156 | 7 | 22.29 | 11.14 | 179 | 8.0 | 0.57 | 2.29 | 2.81 |
| 39 | 2 | 4 | 4 | 2 | 2.00 | 1.00 | 7 | 2.0 | 0.50 | 2.50 | 0.81 |
| 40 | *a*8 | 10 | 40 | 5 | 8.00 | 4.00 | 61 | 7.0 | 0.70 | 2.20 | 2.52 |
| 41 | 45 | 20 | 62 | 10 | 6.20 | 3.10 | 94 | 8.0 | 0.40 | 5.00 | 2.50 |
| 42 | 47 | 12 | 41 | 6 | 6.83 | 3.42 | 64 | 6.0 | 0.50 | 2.33 | 2.46 |
| 43 | 51 | 22 | 22 | 11 | 2.00 | 1.00 | 32 | 2.0 | 0.09 | 6.55 | 0.99 |
| 44 | 60 | 18 | 62 | 9 | 6.89 | 3.44 | 108 | 12.0 | 0.67 | 1.78 | 3.46 |
| 45 | 61 | 22 | 75 | 11 | 6.82 | 3.41 | 112 | 12.0 | 0.55 | 2.45 | 3.37 |
| 46 | 62 | 24 | 124 | 12 | 10.33 | 5.17 | 162 | 9.0 | 0.38 | 3.67 | 2.95 |
| 47 | 30 | 12 | 41 | 6 | 6.83 | 3.42 | 73 | 5.0 | 0.42 | 3.17 | 2.12 |
| 48 | 7 | 5 | 18 | 2 | 9.00 | 3.60 | 81 | 4.0 | 0.80 | 1.40 | 1.92 |
| 49 | 36 | 14 | 96 | 7 | 13.71 | 6.86 | 118 | 5.0 | 0.36 | 3.14 | 2.22 |
| 50 | 69 | 42 | 72 | 21 | 3.43 | 1.71 | 134 | 7.0 | 0.17 | 13.00 | 2.00 |
| 51 | 64 | 34 | 49 | 17 | 2.88 | 1.44 | 83 | 3.0 | 0.09 | 17.47 | 1.21 |
| 52 | 48 | 10 | 34 | 5 | 6.80 | 3.40 | 65 | 6.0 | 0.60 | 2.40 | 2.32 |
| 53 | 32 | 12 | 118 | 6 | 19.67 | 9.83 | 164 | 4.0 | 0.33 | 3.50 | 1.89 |
| 54 | 66 | 28 | 342 | 14 | 24.43 | 12.21 | 402 | 17.0 | 0.61 | 2.07 | 3.92 |
| 55 | 33 | 8 | 28 | 4 | 7.00 | 3.50 | 54 | 5.0 | 0.63 | 2.00 | 2.16 |
| 56 | 16 | 10 | 27 | 5 | 5.40 | 2.70 | 37 | 4.0 | 0.40 | 3.60 | 1.69 |

References for these documents are given in Table 4

*Natural scale for the metrics:* The natural scale for our metrics can be investigated through the measurement theory. Our DW (WSDL) metric is found to be additive in nature (please see the Weyuker's Property 9 and Table 1), and therefore is applicable to the admissible transformation for the ratio scale – the most desirable property of complexity measure. Other metrics are on the interval scale.




doi: 10.1049/iet-sen.2010.0089

**Table 4** Web links for WSDL

| REF | Web link |
|---|---|
| 1 | www.thomas-bayer.com/axis2/services/BLZService?wsdl |
| 2 | www.elguille.info/NET/WebServices/HolaMundoWebS.asmx?WSDL |
| 3 | sdpws.strikeiron.com/SDPv1?WSDL |
| 4 | in2test.lsi.uniovi.es/sqlmutationws?WSDL |
| 5 | ws.strikeiron.com/InnerGears/CityStateByZip2?WSDL |
| 6 | sws-challenge.org/shipper/runner |
| 7 | www.retsinfo.dk/APIS_ |
| 8 | tripleasp.net/Services/ShowCode.asmx?WSDL |
| 9 | services.nirvanix.com/ws/Authentication.asmx?WSDL |
| 10 | service.ecocoma.com/shipping/fedex.asmx?WSDL |
| 11 | www.wubingstudy.com/WebService/Messages.asmx?WSDL |
| 12 | www.iperformonline.com/WebServices/employee/AddUpdateEmployee.asmx?WSDL |
| 13 | webservices.daelab.net/temperatureconversions/TemperatureConversions.wso?WSDL |
| 14 | rangiroa.essi.fr:8080/dotnet/evaluation-cours/EvaluationWS.asmx?WSDL |
| 15 | quisque.com/fr/chasses/blasons/search.asmx?WSDL |
| 15 | webservices.freshegg.com/resources/service1.asmx?WSDL |
| 17 | www.billyclark.com/DesktopModules/FotoVisionDNN/PhotoService.asmx?WSDL |
| 18 | gw1.aql.com/soap/sendsmsservice.php?wsdl |
| 19 | www.devhood.com/services/timelog/timelog-service.asmx?WSDL |
| 20 | services.test.musiccue.net/rapidcueapplication/SecurityProvider.asmx?WSDL |
| 21 | soap1.hopewiser.com:8003 |
| 22 | trial.serviceobjects.com/ce/CurrencyExchange.asmx?WSDL |
| 23 | www.filigris.com/products/docflex_xml/xsddoc/examples/html/eclipse_uml2/index.html |
| 24 | www.oasis-open.org/committees/regrep/documents/2.0/schema/rs.xsd |
| 25 | www.mathertel.de/AJAXEngine/S02_AJAXCoreSamples/CalcService.asmx?WSDL |
| 26 | services.test.musiccue.net/rapidcueapplication/WorkManager.asmx?WSDL |
| 27 | services.test.musiccue.net/rapidcueapplication/SubmissionManager.asmx?WSDL |
| 28 | www.multispeak.org/interface/30j/10_OA_EA.asmx?WSDL |
| 29 | knucklehead.cs.umb.edu/vsowmya/lantetwebservice.asmx?WSDL |
| 30 | coolcampus.csse.monash.edu.au/MonashLibrary/LibraryMaps.asmx?WSDL |
| 31 | www.secureattachment.com/webservices/sadownload.asmx?WSDL |
| 32 | network.grandcentral.com/services/users/docliteral-v1 |
| 33 | terraserver-usa.com/LandmarkService.asmx?WSDL |
| 34 | del.eterio.us/blog/editposts.asmx?WSDL |
| 35 | www.cts.com.pl/webservices/rt_info.asmx?WSDL |
| 36 | www.naf.no/loginservice/main.asmx?WSDL |
| 37 | www.geoservicios.com/V2.0/sgeo/sgeo.asmx?WSDL |
| 38 | itplaza.jeju.go.kr/rpt_ws/Rpt_Ws_FD.asmx?WSDL |
| 39 | demo.soapam.com/services/FedEpayDirectory/FedEpayDirectoryService |
| 40 | itplaza.jeju.go.kr/itplazaweatherservice/ItplazaWeatherService.asmx?WSDL |
| 41 | www.inphoto.cz/Service/PhotoServer.asmx?WSDL |
| 42 | www.filigris.com/products/docflex_xml/xsddoc/examples/html/eclipse_uml2/index.html |
| 43 | ws.strikeiron.com/BusinessDataAppend?WSDL |
| 44 | www.esendex.co.uk/secure/messenger/soap/InboxService.asmx?WSDL |
| 45 | www.sipeaa.it/wset/ServiceET.asmx?WSDL |
| 46 | www.oorsprong.org/websamples.arendsoog/ArendsoogbooksService.wso?WSDL |
| 47 | svc.exaphoto.com/eXaPhoto/CollectionServices.asmx?WSDL |
| 48 | ws.strikeiron.com/MidnightTraderFinancialNews?WSDL |
| 49 | www.simulation.fr/seq/SaintEtiQ.asmx?WSDL |
| 50 | pc218.cgk.affrc.go.jp/PMTypeService/MainEntry.asmx?WSDL |
| 51 | metalmaker.net/metalmaker.asmx?WSDL |
| 52 | localhost/ogsa/services/GLCProcessService |
| 53 | ws.strikeiron.com/ReverseResidentialLookup?WSDL |
| 54 | acims9.acims.arizona.edu/PublicationDB/DEVSPubs.asmx?WSDL |
| 55 | ws.xwebservices.com/XWebBlog/V2/XWebBlog.wsdl |
| 56 | www.saiasecure.com/webservice/shipment/soap.asmx?WSDL |

*Relationship between the attribute to the metric value:* All proposed metrics are the predictor of quality of web services. We explain this point more clearly in the next section.

*Natural and foreseeable side effects of using the instrument:* There is no any side effect of using instrument, because once we develop the complexity calculator, it would be very easy to measure all the complexity measures, without any extra effort and additional workload of manpower of the company. The only cost will be as a result of automation.

## 6 Summary

All the proposed metrics can be used to control the complexity of web services. With increasing complexity in web service, flexibility decreases and, in turn, maintenance of the web service becomes a challenging job. This is because, when a new web process arises by the integration of various applications, the interoperability between these applications becomes main concern for the developers. In this aspect, the degree in flexibility of web services affect the overall integration process and should be managed effectively. Further, the data flow between the participants should be properly managed to provide consistency in data exchanged between the applications via messaging. Hence, as the number of applications that are exposed as a web service increases, the management of data flow between services becomes a major issue that can affect the overall quality of the resulting web process. All the proposed metrics are an attempt to address these issues. For example, if our first metric DW (WSDL) value increases, the web service quality will decrease since it implies inefficient use of memory and time. Similarly, for a given web service having lower DMR implies that the service has a lower data complexity since the messages of the operations are consistent in structures it is easier to remember similar-structured messages. For third and fourth metrics, higher MRS value indicates lower data complexity of WSDL, the higher ME metric value indicates higher data complexity. In other words, higher MRS and lower ME show that the developer makes less effort to understand the messages structures owing to repetition of similar messages.

When we compare all proposed metrics with other metrics, in summary we find the following observations. Reconsider the two figures in Section 3, we can easily observe that APO and OPS metrics cannot capture the differences between those services where data structure of the arguments are different. Similarly, DAR and ARS fail to differentiate these two services since both DAR and ARS metrics do not consider the complexities of data types owing to their internal architectures. Further, the DIT metric for the first service does not take into account the complexities of simple typed arguments whose types are derived by list or union methods of W3C XML schema, for the second service the NOC metric ignores the complexities of the arguments' data types owing to the variety in the contents of the complex types defined in XSD. We can also obtain the same observations from the empirical validation analysis of each metric in Section 4. All these drawbacks of the above metrics are covered by our metrics. Our first metric DW (DSWL) is based on the fact that the arguments may have different data type structures which are expressed by the components of XSD and hence may require different effort to be understood based on internal complexities of these data structures. This is the reason as to why the DW provides the different values for all those services, whose APO and OPS metric values are same. DMR metric is based on the effects of similar-structured messages. DMR metric differentiate to those WSDL, where DW fails to do it (Fig. 7). ME metric is also based on the similarity of the structure of messages, but it is more sensitive than DMR, that is, ME metric can differentiate to those services whose DMR values are same (Fig. 9). Our last metric MRS metric is inversely proportional to ME metric (Fig. 12). The common thing in DMR, ME and MRS is that all of them are based on the similarity of structures of the messages.

We have also evaluated all proposed metrics against Weyuker's set of measurement principles. All the metrics satisfy eight Weyuker's properties except for DMR, which satisfy seven properties. It is worth mentioning that it is not necessary to satisfy all Weyuker's properties. In this point of view, satisfying eight Weyuker's properties by DW, ME, MRRS and seven by DMR metrics proves the robustness of the metrics from theoretical evaluation criteria. Furthermore, the evaluation through Caner's framework proves that all the proposed metrics are developed on the scientific principles and proves the practical utility and worth of the metrics.

It is important to note that there are several factors that may affect the quality of web services. In this paper we are only confined to control the data quality.

## 7 Concluding remarks and future work

We have presented a suite of metrics to evaluate and maintain the quality of the XML web service. All developed metrics have been demonstrated with examples and supported by comparison with the other similar metrics applied on WSDL/web services. Further, to verify our presented metrics and to prove their usefulness and practical applicability we checked them through theoretical and empirical validations. We evaluated all metrics through Weyuker's properties and examined against the practical framework as a part of theoretical validation process. A rigorous empirical validation has been carried out for all proposed metrics. The theoretical, practical and empirical validation proves the worth of the proposed metrics.

Assignments of the upper and lower boundaries of the complexity values for our metrics are aimed as future work. The proposed metrics should be studied in the light of making improvements to the remaining features of web services. The development of an automated tool for computing the metrics is also a task of future work.

## 8 References


1 Cerami, E.: 'web services essentials, distributed applications with XML-RPC, SOAP, UDDI & WSDL' (O'Reilly Publishers, 2002)
2 Erl, T.: 'Service-oriented architecture: a field guide to integrating xml and web services' (Prentice Hall Publishers, 2004)
3 Gourley, D., Totty, B.: 'HTTP: the definitive guide' (O'Reilly Publischers, 2002)
4 Newcomer, E., Lomow, G.: 'Understanding SOA with web-services' (Addison Wesley Professional, 2004)
5 Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: 'Web services platform architecture: soap, wsdl, ws-policy, ws-addressing, ws-bpel, ws-reliable messaging, and more' (Prentice Hall Publishers, 2005)
6 http://www.oasis-open.org/home/index.php, last accessed 21 September 2010
7 http://www.oasis-open.org/specs/index.php#uddiv2, last accessed 21 September 2010
8 http://xml.coverpages.org/uddi.html, last accessed 21 September 2010
9 http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm, last accessed 21 September 2010
10 http://www.w3.org/TR/1998/REC-xml-19980210, last accessed 21 September 2010
11 http://www.w3.org/TR/soap/, last accessed 21 September 2010
12 http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm, last accessed 21 September 2010
13 http://www.w3.org/TR/wsdl, last accessed 21 September 2010
14 Basic, D., Misra, S.: 'Data complexity metrics for web-services', *Adv. Electn. Comput. Eng.*, 2009, **9**, (2), pp. 9–15
15 http://www.w3.org/2002/ws/, last accessed 21 September 2010
16 http://www.w3.org/TR/soap/, last accessed 21 September 2010
17 http://www.w3.org/TR/2004/WD-wsdl20-20040803/#migration, last accessed 21 September 2010





doi: 10.1049/iet-sen.2010.0089

18 WS-I Basic Profile Version 1.0, available at http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html, last accessed 21 September 2010
19 http://www.w3.org/TR/xmlschema-1/, last accessed 21 July 2009
20 http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/, last accessed 21 September 2010
21 http://www.w3.org/TR/2001/PR-xmlschema-0-20010330/, last accessed 21 September 2010
22 http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/, last accessed 21 September 2010
23 ISO/IEC: 'Information technology – text and office systems – regular language description for XML (RELAX) – part 1: RELAX Core, 2000.D TR 22250-1'. Int. Organisation for Standards, 2000
24 http://www.xml.gr.jp/relax, last accessed 21 September 2010
25 Yu, Y., Lu, J., Fernandez-Ramil, J., Yuan, P.: 'Comparing web services with other software components'. Proc. IEEE Int. Conf. on Web Services (ICWS 007), 2007, pp. 388–397
26 Boxall, M., Araban, S.: 'Interface metrics for reusability analysis of components'. Proc. Australian Software Engineering Conf. (ASWEC'04), 2004, pp. 40–51
27 Basic, D., Misra, S.: 'Measuring and evaluating a design complexity metric for XML schema documents', *J. Inf. Sci. Eng.*, 2009, **25**, (5), pp. 1405–1425
28 Davis, J., LeBlanc, R.: 'A study of the applicability of complexity measures', *IEEE Trans. Softw. Eng.*, 1988, **14**, pp. 366–372
29 Shannon, C.E., Weaver, W.: 'The mathematical theory of communication' (University of Illinois Press, Urbana, IL, 1949)
30 Kaner, C.: 'Software engineering metrics: what do they measure and how do we know?'. Proc. Tenth Int. Software Metrics Symp., Metrics, 2004, pp. 1–10
31 Kaner, C.: 'Rethinking software metrics: evaluating measurement schemes', *Softw. Test. Qual. Eng.*, 2000, **2**, (2), pp. 50–57
32 Briand, L.C., Morasca, S., Basily, V.R.: 'Property based software engineering measurement', *IEEE Trans. Softw. Eng.*, 1996, **22**, (1), pp. 68–86
33 Weyuker, E.: 'Evaluating software complexity measures', *IEEE Trans. Softw. Eng.*, 1998, **14**, pp. 1357–1365
34 Fenton, N.: 'New software quality metrics methodology standards fills measurement needs', *IEEE Comput.*, 1993, **26**, (4), pp. 105–106
35 Fenton, N.: 'Software measurement: a necessary scientific basis', *IEEE Trans. Softw. Eng.*, 1994, **20**, (3), pp. 199–206
36 IEEE Computer Society: 'Standard for software quality metrics methodology'. Revision IEEE Standard, 1998, pp. 1061–1998
37 Kitchenham, B., Pfleeger, S.L., Fenton, N.: 'Towards a framework for software measurement validation', *IEEE Trans. Softw. Eng.*, 1995, **21**, (12), pp. 929–943
38 Haug, M., Olsen, E.W., Consolini, L., Bergman, L.: 'Software process improvement metrics, measurement and process modelling' (Springer, 2001)
39 Zuse, H.: 'Software complexity measures and methods' (Walter de Gruyter, Berline, 1991)
40 Zuse, H.: 'Properties of software measures', *Softw. Qual. J.*, 1992, **1**, pp. 225– 260
41 Cherniavsky, J.C., Smith, C.H.: 'On Weyuker's axioms for software complexity measures', *IEEE Trans. Softw. Eng.*, 1991, **17**, pp. 636–638
42 Fenton, N.E.: 'Software metrics – a rigorous approach' (Chapman & Hall, London, 1991)
43 Misra, S., Akman, I.: 'Applicability of Weyuker's properties on OO metrics some misunderstandings', *J. Comput. Inf. Sci.*, 2008, **15**, (1), pp. 17–24
44 Gursaran, G.R.: 'On the applicability of Weyuker property nine to object oriented structural inheritance complexity metrics', *IEEE Trans. Softw. Eng.*, 2001, **27**, (4), pp. 361–364
45 Sharma, N., Joshi, P., Joshi, R.K.: 'Applicability of Weyuker's property 9 to object oriented metrics', *IEEE Trans. Softw. Eng.*, 2006, **32**, (3), pp. 209–211
46 Zhang, L., Xie, D.: 'Comments on "on the applicability of Weyuker property nine to object oriented structural inheritance complexity metrics" ', *IEEE Trans. Softw. Eng.*, 2002, **28**, (5), pp. 526–527
47 Chhabra, J.K., Gupta, V.: 'Evaluation of object-oriented spatial complexity measures', *SIGSOFT Softw. Eng. Notes*, 2009, **34**, pp. 1–5
48 Cardoso, J.: 'Control flow complexity measurement of process and Weyuker's properties', *Trans. Enform. Syst. Sci. Eng.*, 2005, **8**, pp. 213–218
49 Misra, S., Akman, I.: 'Weighted class complexity: a measure of complexity for object oriented systems', *J. Inf. Sci. Eng.*, 2008, **24**, pp. 1689–1708
50 Auprasert, B., Limpiyakorn, Y.: 'Towards structured software cognitive complexity measurement with Granular Computing strategies'. Proc. Eighth IEEE Int. Conf. on Cognitive Informatics, 2009, pp. 365–370
51 Aggarwal, K.K., Singh, Y., Kaur, A., Melhotra, R.: 'Software design metrics for object orinted software', *J. Object Technol.*, 2006, **6**, (1), pp. 121–138

## 9 Appendix

```
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:tns="com.esendex.ems.soapinterface" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="com.esendex.ems.soapinterface">
 <wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="com.esendex.ems.soapinterface">
   <s:element name="GetMessages">
    </s:complexType>
   </s:element>
   <s:element name="GetMessagesResponse">
    <s:complexType>
     <s:sequence>
  <s:element maxOccurs="1" minOccurs="0" name="GetMessagesResult" type="tns:ArrayOfMessage"/>
     </s:sequence>
    </s:complexType>
   </s:element>
   <s:complexType name="ArrayOfMessage">
    <s:sequence>
     <s:element maxOccurs="unbounded" minOccurs="0" name="message" nillable="true" type="tns:message"/>
    </s:sequence>
   </s:complexType>
   <s:complexType name="message">
    <s:sequence>
     <s:element maxOccurs="1" minOccurs="0" name="id" type="s:string"/>
     <s:element maxOccurs="1" minOccurs="0" name="originator" type="s:string"/>
     <s:element maxOccurs="1" minOccurs="0" name="recipient" type="s:string"/>
     <s:element maxOccurs="1" minOccurs="0" name="body" type="s:string"/>
     <s:element maxOccurs="1" minOccurs="1" name="sentat" type="s:dateTime"/>
     <s:element maxOccurs="1" minOccurs="1" name="receivedat" type="s:dateTime"/>
     <s:element maxOccurs="1" minOccurs="1" name="type" type="tns:MessageType"/>
     <s:element maxOccurs="1" minOccurs="1" name="status"   tpe="tns:MessageStatusCode"/>
     <s:element maxOccurs="1" minOccurs="0" name="username" type="s:string"/>
    </s:sequence>
   </s:complexType>
   <s:simpleType name="MessageType">
    <s:restriction base="s:string">
     <s:enumeration value="Text"/>
     <s:enumeration value="Binary"/>
     <s:enumeration value="SmartMessage"/>
     <s:enumeration value="Unicode"/>
    </s:restriction>
   </s:simpleType>
   <s:simpleType name="MessageStatusCode">
    <s:restriction base="s:string">
     <s:enumeration value="Queued"/>
     <s:enumeration value="Sent"/>
     <s:enumeration value="Delivered"/>
     <s:enumeration value="Failed"/>
    </s:restriction>
   </s:simpleType>
   <s:element name="MessengerHeader" type="tns:MessengerHeader"/>
   <s:complexType name="MessengerHeader">
    <s:sequence>
     <s:element maxOccurs="1" minOccurs="0" name="Username" type="s:string"/>
     <s:element maxOccurs="1" minOccurs="0" name="Password" type="s:string"/>
     <s:element maxOccurs="1" minOccurs="0" name="Account" type="s:string"/>
    </s:sequence>
   </s:complexType>
   <s:element name="GetMessageByID">
    <s:complexType>
     <s:sequence>
      <s:element maxOccurs="1" minOccurs="0" name="id" type="s:string"/>
     </s:sequence>
    </s:complexType>
   </s:element>
   <s:element name="GetMessageByIDResponse">
    <s:complexType>
     <s:sequence>
      <s:element maxOccurs="1" minOccurs="0" name="GetMessageByIDResult" type="tns:message"/>
     </s:sequence>
    </s:complexType>
   </s:element>
   <s:element name="GetMessagesByID">
    <s:complexType>
     <s:sequence>
      <s:element maxOccurs="1" minOccurs="0" name="ids" type="tns:ArrayOfString"/>
     </s:sequence>
```

**Fig. 18** *Real-life example: WSDL documents*




doi: 10.1049/iet-sen.2010.0089

```
      </s:complexType>
     </s:element>
     <s:complexType name="ArrayOfString">
      <s:sequence>
       <s:element maxOccurs="unbounded" minOccurs="0" name="string" nillable="true"
type="s:string"/>       </s:sequence>
     </s:complexType>
     <s:element name="GetMessagesByIDResponse">
      <s:complexType>
       <s:sequence>
        <s:element maxOccurs="1" minOccurs="0" name="GetMessagesByIDResult" type="tns:ArrayOfMessage"/>
       </s:sequence>
      </s:complexType>
     </s:element>
     <s:element name="GetMessagesForDay">
      <s:complexType>
       <s:sequence>
        <s:element maxOccurs="1" minOccurs="1" name="year" type="s:int"/>
        <s:element maxOccurs="1" minOccurs="1" name="month" type="s:int"/>
        <s:element maxOccurs="1" minOccurs="1" name="day" type="s:int"/>
       </s:sequence>
      </s:complexType>
     </s:element>
     <s:element name="GetMessagesForDayResponse">
      <s:complexType>
       <s:sequence>
        <s:element maxOccurs="1" minOccurs="0" name="GetMessagesForDayResult" type="tns:ArrayOfMessage"/>
       </s:sequence>
      </s:complexType>
     </s:element>
     <s:element name="GetMessagesForDateRange">
      <s:complexType>
       <s:sequence>
        <s:element maxOccurs="1" minOccurs="1" name="startDate" type="s:dateTime"/>
        <s:element maxOccurs="1" minOccurs="1" name="endDate" type="s:dateTime"/>
       </s:sequence>
      </s:complexType>
     </s:element>
     <s:element name="GetMessagesForDateRangeResponse">
      <s:complexType>
       <s:sequence>
        <s:element maxOccurs="1" minOccurs="0" name="GetMessagesForDateRangeResult"
type="tns:ArrayOfMessage"/>
       </s:sequence>
      </s:complexType>     </s:element>
     <s:element name="DeleteMessage">
      <s:complexType>
       <s:sequence>
        <s:element maxOccurs="1" minOccurs="0" name="id" type="s:string"/>
       </s:sequence>
      </s:complexType>
     </s:element>
     <s:element name="DeleteMessageResponse">
      <s:complexType></s:complexType>
     </s:element>
     <s:element name="DeleteMessages">
      <s:complexType>
       <s:sequence>
        <s:element maxOccurs="1" minOccurs="0" name="ids" type="tns:ArrayOfString"/>
       </s:sequence>
      </s:complexType>
     </s:element>
     <s:element name="DeleteMessagesResponse">
      <s:complexType></s:complexType>
     </s:element>
    </s:schema>
   </wsdl:types>
   <wsdl:message name="GetMessagesSoapIn"><!-- G1(1,1)-->
    <wsdl:part element="tns:GetMessages" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="GetMessagesSoapOut"><!--G2(14,9)-->
    <wsdl:part element="tns:GetMessagesResponse" name="parameters"/>
   </wsdl:message>
   <wsdl:message name="GetMessagesMessengerHeader"><!--G3(4,3)-->
    <wsdl:part element="tns:MessengerHeader" name="MessengerHeader"/>
   </wsdl:message>
  <wsdl:message name="GetMessagesByIDSoapIn"><!--G6(3,1)-->
   <wsdl:part element="tns:GetMessagesByID" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="GetMessagesByIDSoapOut"><!--G2(14,9)-->
   <wsdl:part element="tns:GetMessagesByIDResponse" name="parameters"/>
```

**Fig. 18** *Continued*

```
  </wsdl:message>
  <wsdl:message name="GetMessagesMessengerHeader"><!--G3(4,3)-->
    <wsdl:part element="tns:MessengerHeader" name="MessengerHeader"/>
  </wsdl:message>
<wsdl:message name="GetMessagesByIDSoapIn"><!--G6(3,1)-->
  <wsdl:part element="tns:GetMessagesByID" name="parameters"/>
</wsdl:message>
<wsdl:message name="GetMessagesByIDSoapOut"><!--G2(14,9)-->
  <wsdl:part element="tns:GetMessagesByIDResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="GetMessagesByIDMessengerHeader"><!--G3(4,3)-->
  <wsdl:part element="tns:MessengerHeader" name="MessengerHeader"/>
</wsdl:message>
<wsdl:message name="GetMessagesForDaySoapIn"><!--G3(4,3)-->
  <wsdl:part element="tns:GetMessagesForDay" name="parameters"/>
</wsdl:message>
<wsdl:message name="GetMessagesForDaySoapOut"><!--G2(14,9)-->
  <wsdl:part element="tns:GetMessagesForDayResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="GetMessagesForDayMessengerHeader"><!--G3(4,3)-->
  <wsdl:part element="tns:MessengerHeader" name="MessengerHeader"/>
</wsdl:message>
<wsdl:message name="GetMessagesForDateRangeSoapIn"><!--G7(3,2)-->
  <wsdl:part element="tns:GetMessagesForDateRange" name="parameters"/>
</wsdl:message>
<wsdl:message name="GetMessagesForDateRangeSoapOut"><!--G2(14,9)-->
  <wsdl:part element="tns:GetMessagesForDateRangeResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="GetMessagesForDateRangeMessengerHeader"><!--G3(4,3)-->
  <wsdl:part element="tns:MessengerHeader" name="MessengerHeader"/>
</wsdl:message>
<wsdl:message name="DeleteMessageSoapIn"><!--G4(2,1)-->
  <wsdl:part element="tns:DeleteMessage" name="parameters"/>
</wsdl:message>
<wsdl:message name="DeleteMessageSoapOut"><!- -G1(1,1)-->
  <wsdl:part element="tns:DeleteMessageResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="DeleteMessageMessengerHeader"><!--G3(4,3)-->
  <wsdl:part element="tns:MessengerHeader" name="MessengerHeader"/>
</wsdl:message>
<wsdl:message name="DeleteMessagesSoapIn"><!--G6(3,1)-->
  <wsdl:part element="tns:DeleteMessages" name="parameters"/>
</wsdl:message>
<wsdl:message name="DeleteMessagesSoapOut"><!-- G1(1,1)-->
  <wsdl:part element="tns:DeleteMessagesResponse" name="parameters"/>
</wsdl:message>
<wsdl:message name="DeleteMessagesMessengerHeader"><!--G3(4,3)-->
  <wsdl:part element="tns:MessengerHeader" name="MessengerHeader"/>
</wsdl:message>
<wsdl:portType name="InboxServiceSoap">
  <wsdl:operation name="GetMessages">
    <wsdl:input message="tns:GetMessagesSoapIn"/>
    <wsdl:output message="tns:GetMessagesSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="GetMessageByID">
    <wsdl:input message="tns:GetMessageByIDSoapIn"/>
    <wsdl:output message="tns:GetMessageByIDSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="GetMessagesByID">
    <wsdl:input message="tns:GetMessagesByIDSoapIn"/>
    <wsdl:output message="tns:GetMessagesByIDSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="GetMessagesForDay">
    <wsdl:input message="tns:GetMessagesForDaySoapIn"/>
    <wsdl:output message="tns:GetMessagesForDaySoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="GetMessagesForDateRange">
    <wsdl:input message="tns:GetMessagesForDateRangeSoapIn"/>
    <wsdl:output message="tns:GetMessagesForDateRangeSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="DeleteMessage">
    <wsdl:input message="tns:DeleteMessageSoapIn"/>
    <wsdl:output message="tns:DeleteMessageSoapOut"/>
  </wsdl:operation>
  <wsdl:operation name="DeleteMessages">
    <wsdl:input message="tns:DeleteMessagesSoapIn"/>
    <wsdl:output message="tns:DeleteMessagesSoapOut"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="InboxServiceSoap" type="tns:InboxServiceSoap">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
```

**Fig. 18** *Continued*




doi: 10.1049/iet-sen.2010.0089

```
<wsdl:operation name="GetMessages">
  <soap:operation soapAction="com.esendex.ems.soapinterface/GetMessages" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
    <soap:header message="tns:GetMessagesMessengerHeader" part="MessengerHeader" use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetMessageByID">
  <soap:operation soapAction="com.esendex.ems.soapinterface/GetMessageByID" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
    <soap:header message="tns:GetMessageByIDMessengerHeader" part="MessengerHeader" use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetMessagesByID">
  <soap:operation soapAction="com.esendex.ems.soapinterface/GetMessagesByID" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
    <soap:header message="tns:GetMessagesByIDMessengerHeader" part="MessengerHeader" use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetMessagesForDay">
  <soap:operation soapAction="com.esendex.ems.soapinterface/GetMessagesForDay" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
    <soap:header message="tns:GetMessagesForDayMessengerHeader" part="MessengerHeader" use="literal"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetMessagesForDateRange">
  <soap:operation soapAction="com.esendex.ems.soapinterface/GetMessagesForDateRange" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
    <soap:header message="tns:GetMessagesForDateRangeMessengerHeader" part="MessengerHeader" use="liter"/>
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="DeleteMessage">
  <soap:operation soapAction="com.esendex.ems.soapinterface/DeleteMessage" style="document"/>
  <wsdl:input>
    <soap:body use="literal"/>
      <soap:header message="tns:DeleteMessageMessengerHeader" part="MessengerHeader" use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="DeleteMessages">
    <soap:operation soapAction="com.esendex.ems.soapinterface/DeleteMessages" style="document"/>
    <wsdl:input>
      <soap:body use="literal"/>
      <soap:header message="tns:DeleteMessagesMessengerHeader" part="MessengerHeader" use="literal"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="InboxService">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Manage incoming messages queued in the account inbox.Click
here for a &lt;a href="https://www.esendex.com/secure/registration/evaluation.aspx" title="SMS Service Trial"&gt;free
trial&lt;/a&gt;.</documentation>
  <port xmlns="http://schemas.xmlsoap.org/wsdl/" binding="tns:InboxServiceSoap" name="InboxServiceSoap">
    <soap:address location="http://www.esendex.co.uk/secure/messenger/soap/InboxService.asmx"/>
  </port>
</wsdl:service>
</wsdl:definitions>
```

**Fig. 18** *Continued*