

Measuring and Evaluating a Design Complexity Metric for XML Schema Documents

DILEK BASCI AND SANJAY MISRA

Department of Computer Engineering

Atilim University

Ankara, Turkey

The eXtensible Markup Language (XML) has been gaining extraordinary acceptance from many diverse enterprise software companies for their object repositories, data interchange, and development tools. Further, many different domains, organizations and content providers have been publishing and exchanging information via internet by the usage of XML and standard schemas. Efficient implementation of XML in these domains requires well designed XML schemas. In this point of view, design of XML schemas plays an extremely important role in software development process and needs to be quantified for ease of maintainability. In this paper, an attempt has been made to evaluate the quality of XML schema documents (XSD) written in W3C XML Schema language. We propose a metric, which measures the complexity due to the internal architecture of XSD components, and due to recursion. This is the single metric, which cover all major factors responsible for complexity of XSD. The metric has been empirically and theoretically validated, demonstrated with examples and supported by comparison with other well known structure metrics applied on XML schema documents.

Keywords: XML schema metrics, data representation, W3C, SOA, empirical validation, data representation

1. INTRODUCTION

The usage of the Web continues to proliferate at an astounding rate, and the amount of data conveyed by large number of documents on the Web has also exploded increasingly. While information sharing among different parties connected to Internet becomes widespread, the need for information to be transmitted from one party to another has result in the emergence of a standard mechanism for information exchange that can be easily implemented by different domains. The eXtensible Markup Language [1] (XML), has become an increasingly significant part of the IT mainstream since its invention and it has been gaining a general acceptance as a standard for data representation and transmission between distinct applications running on different platforms, such as health-care, manufacturing, financial services, government and publishing sectors. In recent years, Web Services, as a new type of distributed application, use XML documents for their data representation. Data representation architecture represents the foundation layer in service oriented architecture (SOA), and XML establishes the format and structure of messages traveling throughout the services [2]. While information sharing in distributed computing environment becomes widespread, XML, has become more popular as a universal data format for exchanging structured information via internet communications. Due to powerful expressive capability of data and documents, flexible nature and ease of

Received December 14, 2007; accepted March 3, 2008.

Communicated by Jonathan Lee, Wei-Tek Tsai and Yau-Hwang Kuo.

implementation, XML has been gaining extraordinary acceptance from many enterprise, software companies for their object repositories, data interchange, and development tools. As a standard mechanism for structuring data XML has been easily adopted in many diverse fields and creates a great opportunity for better information retrieval.

In XML context, the data representations are made by designing schemata which can be written in different XML schema languages such as DTD [1], W3C XML Schema [3], RELAX [4, 5]. W3C XML Schema [3] and DTDs [1] are the most favored schema languages for generating XML documents. However, deploying XML documents is a challenging problem for an application without using supporting schema technology. The most important tangible aspect of an XML schema is that an XML schema specifies a contract between software applications or between parts of a software application. In large software applications the notion of a contract provides many benefits such as simplifying software modularization, resource allocation, testing, and deployment. Using schemas not only provides common understanding about exchanged data but also the ability of easy access methods for XML documents to be validated. With the successful design and implementation of schemas, the developers can have the capability of increasing productivity, improving software reliability, minimizing development time, and decreasing time to market. Further, well designed schemas provide a way for XML documents to be sorted and retrieved efficiently and effectively. Neglecting schemas implies that the schema validators are not used to determine if a given XML document satisfies desired data transported among applications. In such a case the required check have to be performed by the application programs implying that the application developers have to write lengthy code.

All these considerations imply that schemas play an important role in software construction project and hence, need to be properly designed, so that it can be easily maintained in order for XML data to be effectively and properly used by distributed applications. In order for the schema to be easily maintained, schema metrics must be developed to enable quantification of schema size, complexity, quality and the other properties; however, a few researches [6-11], that deal with schema quality and complexity metric have been done. In our previous work, we proposed a metric for XML schema [12]. It was our initial attempt in this area and in the present work we extended our previous metric by considering complexity due to recursion, an important factor which directly affects the complexity. Further, a newly proposed measure is acceptable, only when its usefulness has been proved by empirical validation process. In the present work, we empirically validated our metric by applying it against 65 real examples.

The common approach to measure the complexity of XML schema documents in [6] and, [8] is to count the number of schema components. However, the metrics that measure schema's complexity by counting the number of each component do not give sufficient information about complexity value of a given schema and the complexity of each independent component is also important, which were neglected in existing metrics. This is the main motivation for us to develop a new metric for XSDs. Another motivation to focus on to develop the complexity metric for XSDs is that W3C XML Schema language [13] has the stronger capability than DTD to describe the vocabularies of XML documents and has general agreement of being the schema language of the future for XML. We suggest that the complexity of a given XML schema document written in W3C XML Schema language closely depends on complexities of internal complexities of its building

components, that is, each component contributes their complexity values on the basis of their design architectures to the schema document's complexity. In this point of view, it will be meaningful to assign a weight value for each component that reflects the effort required to understand the structure of each component called complexity degree. Further, for calculating the complexity of the schema document each of its component's weight values should be summed up in order to evaluate a single complexity value.

The paper is organized in the following way. Our proposed metric is defined in section 2 and demonstrated by real example in section 3. The theoretical, empirical validation of the proposed metric and a comparative study with other measures have been done in the section 4. Lastly, section 5, provides concluding remarks and a reflection on future work.

2. PROPOSED METRIC

Major building components of XML Schema are elements having simple or complex type as a type reference; attributes, simple and complex types, elements and attributes group definitions/declarations [13]. A simple type can be built-in simple type or user-defined simple type derived from the other simple types. The schema document may not necessarily validate any XML document and can be designed as a library document. Based on its design style [14] a given schema may have different number of components declared/defined locally or globally. For example, the number of complex or simple type definitions may be greater than element with or without attributes declaration or vice versa or the schema may use global elements and attributes group definitions or encode all groups inside complex type's content model definition instead. The schema may also contain recursion in which an element can be included within an element of the same type directly or indirectly as a child element. Further, the schema may use components via import, include and redefine mechanism [13-17] of W3C XML Schema from external schema files.

Accordingly, the complexity of XSD depends upon the following factors:

- (a) The complexity due to elements and attributes definitions/declarations.
- (b) The complexity due to elements and attributes group definitions/declarations.
- (c) The complexity due to all types including user defined/built-in simple type and complex type definitions/declarations. Note that, by using type derivation methods (extension or restriction) of XML Schema Language users are enabled to define new types based on their needs.
- (d) The complexity due to elements definitions/declarations that contain recursion. Note that only complex-typed elements tend to contain recursion.
- (e) The complexity due to components that are included or imported from external schema files. These components can be elements, attributes, groups or simple/complex types.

Accordingly the total complexity of the XSD is given by the following formula

$$C(XSD) = C(V_g) + C(G_g) + C(T_g) \quad (1)$$

where, $C(XSD)$: complexity value of the schema document (XSD) written in XML Schema language;

$C(V_g)$: total complexity values of all global elements and attributes that can be included/imported from external XSDs or can be declared/defined in the current XSD.

$C(G_g)$: total complexity values of unreferenced global elements and attributes group that can be declared/defined in the current XSD.

$C(T_g)$: total complexity values of unreferenced global complex and user-defined/built-in simple type definitions/declarations of XML Schema document.

By the word “unreferenced” we mean components that have no reference made within any component definitions of the current schema. The reason for considering unreferenced components is that; since an element or attribute being declared locally or globally can have type reference to any globally defined complex or simple type definitions, we are at risk adding two times both element’s, attribute’s complexity values and their respective type complexity values to $C(XSD)$. Similarly, since global elements and attributes group can be referenced inside any complex type definitions or the other group definitions we again risk for adding two times complexities of both complex type’s and group’s definitions. However, it is not common that a given schema document includes an unreferenced components except for some schema files designed as a library document.

The complexity of the schema document is also affected by the elements that contain recursion. An XML schema is said to be recursive when the type definition of an element in it allow for elements of the same name and type to appear in their own definition. Recursion may be explicit or implicit. Hence, any element that has recursive type definition or recursive child element can be considered as a recursive element. In its internal structure a recursive element may contain non-recursive regions that include number of non-recursive child elements and number of attributes. Hence, each component declared inside the type definition of the recursive element affects the complexity of that recursive element. One may argue that any element having more recursive child element is more complex than any other element having less recursive child elements and than any element that has only non-recursive elements or attributes. It is also arguable that as the number of recursive elements increases in the schema document so is the effort required comprehending the schema document.

In our opinion, it is possible for a given schema document which includes recursive elements, the effort required to understand that schema may be less than a schema that does not contain recursive element. In addition, we also suggest that a schema file having greater number of recursive elements may be less complex than a schema file having less number of recursive elements due to the internal structures of recursive elements.

Definitions of each component of $C(XSD)$ are given below:

$C(V_g)$ can be defined as:

$$C(V_g) = C(E_g) + C(A_g) \quad (2)$$

where, $C(E_g)$ and $C(A_g)$ are complexities of global elements and attributes definitions/declarations respectively and are defined as:

$$C(E_g) = \sum_{i=1}^N w e_i E_{gi}; \quad (3)$$

$$C(A_g) = \sum_{j=1}^M wa_j A_{g_j}; \tag{4}$$

where N, M are the total number of *global* element, attribute declarations; we_i, wa_j are corresponding type definition weight values of element E_{g_i} and attribute A_{g_j} . It is important to note that the weight value of a component reflects its complexity degree.

$C(G_g)$ can be defined as:

$$C(G_g) = C(EG_g) + C(AG_g) \tag{5}$$

where $C(EG_g)$ and $C(AG_g)$ are the complexities of global elements and attributes group definition/declaration respectively and are defined as:

$$C(EG_g) = \sum_{t=1}^K weg_t EG_{g_t} \tag{6}$$

$$C(AG_g) = \sum_{s=1}^P wag_s AG_{g_s} \tag{7}$$

where K, P are the total number of *global unreferenced* elements and attributes group declarations/definitions; weg_t, wag_s are corresponding weight values of elements group EG_{g_t} and attributes group AG_{g_s} respectively.

$C(T_g)$ is defined as:

$$C(T_g) = C(cT_g) + C(sT_g) \tag{8}$$

where $C(cT_g)$ and $C(sT_g)$ are complexities of global complex and simple type definition respectively and are defined as:

$$C(cT_g) = \sum_{r=1}^R wc_r cT_{g_r} \tag{9}$$

$$C(sT_g) = \sum_{q=1}^Q ws_q sT_{g_q} \tag{10}$$

where R, Q are the number of *global unreferenced* complex-type and simple-type definitions; wc_r, ws_q are corresponding weight values of complex and simple type definitions cT_{g_r}, sT_{g_q} respectively. Thus;

$$\begin{aligned} C(XSD) &= C(V_g) + C(G_g) + C(T_g) \\ &= [C(E_g) + C(A_g)] [C(EG_g) + C(AG_g)] + [C(cT_g) + C(sT_g)] \\ &= \left[\sum_{i=1}^N we_i E_{g_i} + \sum_{j=1}^M wa_j A_{g_j} \right] + \left[\sum_{t=1}^K weg_t EG_{g_t} + \sum_{s=1}^P wag_s AG_{g_s} \right] + \\ &\quad \left[\sum_{r=1}^R wc_r cT_{g_r} + \sum_{q=1}^Q ws_q sT_{g_q} \right]. \end{aligned} \tag{11}$$

As explained earlier, weight values for each schema component can reflect the complexity degree of corresponding component and are assigned on the basis of their design structures *i.e.* its internal architectures, since the components of XSDs can be dependent on each other in the sense that the definition/declaration of any component may use the other components [13-15, 17]. As a result, while the weight value of element depends on its type's weight value, that type's weight value depends on its internal structure. In this point of view, due to the complex type definition can include nested compositors or particles with different number of occurrences [13-15, 17] based on its content model, the weight value of an element having simple type as a type reference differs from that of the element having complex type. Similarly, the weight value of a complex type with simple content model may differ from that of a complex type with complex content model. Hence, while assigning weight value to a complex type definition, weight values of each constituent member encoded in the content model of it should be considered. This is also valid to evaluate weight values for the elements and attributes group definitions since each member of any type of group definitions may have different weight values.

We assume that built-in simple types have the weight value of 1 since these types are simplest data type structure used in the schema document (XSD). In the schema document an element type that does not explicitly specify a structure type implicitly specifies *anyType* [13-15, 17] as the structure type. The content of an element in an XML instance whose structure type is *anyType* is unconstrained. The simplest type structure for *anyType* can be a built-in simple type. For this reason we assumed that the weight value for any attributes or elements whose type definition is specified by *anyType* is 1. The `<any>` element provides a mechanism for specifying elements with what the XML Schema Recommendation [15, 16] calls a wildcard. By the usage of the `<any>` element an XML validator validates elements in an XML instance document. The `<any>` element generally specifies a set of namespaces against which the XML validator may validate. The XML validator searches each namespace for global element types that might correspond to the elements referenced in the XML instance. Since, in the simplest case that global element types can be a simple type we made another assumption that the weight for an element declared by `<any>` element in the schema is 1. Similarly, we also assume that the weight value for an attribute declared by `<anyAttribute>` element in the schema is 1 since `<anyAttribute>` element is analogous to the `<any>` element of W3C XML Schema. Another point that needs to pay attention is that we only take into considerations referenced components of the external schemas that are included to the current schema via *import*, *include* and *redefinition* mechanism while evaluating complexity value of the current schema document. Based on these assumptions, weight values for each XML schema component can be calculated as follows:

Element's weight value:

$$we = ws, \text{ if elements have } \textit{simple-type} \quad (12.1)$$

$$= wc, \text{ if element has } \textit{complex-type} \quad (12.2)$$

$$= wcr, \text{ if element is global and has complex type having recursion} \quad (12.3)$$

$$= 1, \text{ if element is declared by using } \textit{<any>} \text{ element} \quad (12.4)$$

$$= 1, \text{ if element is declared by using } \textit{anyType} \quad (12.5)$$

$$= R, \text{ if element has recursion and is local.} \quad (12.6)$$

While calculating the weight value of an element we consider whether that element contains recursion or not. If an element does not contain recursive child elements that element can be inferred as a non-recursive element and its weight value is equal to the value of wc . However if an element has recursive child elements we also take into consideration whether that element is declared globally or locally. If an element is declared globally and contains recursive child element then the weight value for that element is equal to the value of wcr which is given in Eq. (16.2). If an element containing recursion is declared as a child element of any other element or is declared inside any elements group or has a reference to any recursive global element then the weight value for that element is assumed to be equal to the value of the variable R greater than 1. The reason for this assumption is that due to the recursion the weight value for any recursive child element may go infinity since that element may also consist of number of recursive child elements. We assumed that the value for the variable R is greater than 1 since any complex-type is derived from the *anyType* by default and since it is recursive it should have at least one child element which makes that element recursive.

Attribute's weight value:

$$wa = ws, \text{ since attributes can only have simple-type} \quad (13.1)$$

$$= 1, \text{ if attribute is declared by } \langle \text{anyAttribute} \rangle \text{ elements.} \quad (13.2)$$

Weight values of elements group can be calculated based on its definition mechanism that is:

$$wa = \sum_{i=1}^N we_i E_i, \text{ if not redefined} \quad (14.1)$$

$$= we_{g_{baseGroup}} + \sum_{i=1}^N we_i E_i, \text{ if redefined by extension} \quad (14.2)$$

$$= we_{g_{baseGroup}} - \sum_{i=1}^N we_i E_i, \text{ if redefined by restriction.} \quad (14.3)$$

where $we_{g_{baseGroup}}$ is the weight value of base elements group of defined elements group if it is extended or restricted by redefinition mechanism of W3C XML Schema; we_i is the weight value of corresponding declared element E_i in the case where the group is not redefined the weight value of the base group is 0. The capital N in Eq. (15.1) represents the number of elements if the group is not redefined. In Eq. (15.2) the weight values of N number of the newly declared elements inside group redefinition is added to the base group weight value if the group is redefined by extension. In Eq. (15.3) the N number of not inherited elements from base group to redefined group is subtracted from the weight value of the base group if the group is derived by restriction. Note that element and attribute groups can only be derived via redefinition mechanism of W3C XML Schema [14, 17].

Weight values of attributes group can be calculated by summing up all its attributes' weight values and define as:

$$wag = \sum_{i=1}^N wa_i A_i, \text{ if not redefined} \quad (15.1)$$

$$= wag_{baseGroup} + \sum_{i=1}^N wa_i A_i, \text{ if redefined by extension} \quad (15.2)$$

$$= wag_{baseGroup} - \sum_{i=1}^N wa_i A_i, \text{ if redefined by restriction.} \quad (15.3)$$

Here, wag definition is similar to weg definition, but, in this case we are mentioning about attributes.

The weight value of a complex-type can be calculated by summing the weight values of all its constituent components (elements, attributes, and groups) and can be defined as:

$$wc = wc_{baseType} \pm \left[\sum_{i=1}^N we_i E_i + \sum_{j=1}^M wa_j A_j + \sum_{t=1}^K weg_t EG_t + \sum_{s=1}^P wag_s AG_s \right] \quad (16.1)$$

where $wc_{baseType}$ is the weight value of complex-type's parent if it is explicitly derived; we_i , wa_j , weg_t , wag_s are corresponding weight values of element E_i , attribute A_j , element group EG_t and attribute group AG_s respectively; N , M , K , P are the number of local or referenced elements, attributes, element groups and attribute groups definitions/declarations respectively. A complex type can be derived from the simple types or the other complex types by restriction or extension mechanism. If derivation method is not explicitly specified we assumed that the base type weight value of a complex type is 1 since a complex type is derived from *anyType* type by restriction, by default. If a complex type is derived by restriction the capitals N , M , K , P represent the number of corresponding components that are not inherited from base type and the weight values of all these components are subtracted from the weight value of the base type. In the case where a complex type is derived by extension, the capitals N , M , K , P represent the number of corresponding components that are newly inserted and the weight values of all these components to derived complex-type definition are added to the base type weight value.

The weight value of a complex-type that contains recursion can be calculated in a similar way that the value for wc is calculated. However, in a recursive complex type case we consider the number of recursive branches (child elements) and the complexity values of all elements and attributes that are fall into non-recursive regions. Note that, by non-recursive region we refer the region that are out of the recursive branches.

$$wcr = wcr_{baseType} \pm \left[\sum_{i=1}^N we_i E_i + \sum_{j=1}^M wa_j A_j + \sum_{t=1}^K weg_t EG_t + \sum_{s=1}^P wag_s AG_s \right] + NRC * R \quad (16.2)$$

where $wcr_{baseType}$ is the weight value of recursive complex-type's parent; based on the derivation method of the complex type, the capitals N , M , K , P , are the number of not inherited or newly inserted local or referenced elements, attributes, element groups and attribute groups definitions/declarations that are fall into non-recursive region respectively; NRC is the total number of child elements that contain recursion; R is the any

positive integer number greater than 1. Note that while calculating the value of the complex type, wcr that contains recursive child (branches) the weight values of its recursive descendents are not considered due to the recursion.

Weight value of a simple-type can be defined as:

$$ws = 1, \text{ if it is built in simple type} \quad (17.1)$$

$$= r | r \text{ is the number of restriction, if it is derived by restriction.} \quad (17.2)$$

$$= u | u = \sum_{i=1}^P w_i M_i, \text{ if it is derived by union.} \quad (17.3)$$

$$= l | l \text{ is the weight value of item type, if it is derived by list.} \quad (17.4)$$

In Eq. (17.3), P is the number of members declared within union simple-type; w_i is the weight value of member M_i that can be built-in or derived simple type and equals to ws since the types of the members should only be simple type [13-15, 17].

3. ILLUSTRATION OF PROPOSED METRIC

We have demonstrated our metric by taking an example *exampleXMLSchema.xsd* document as given in Fig. 1. We calculated the weight values of each component declared/defined inside the example schema given in Fig. 1, and are shown in Table 1. While calculating the complexity value for the example schema document we sum all the weight values of all global elements, attributes, that of all unreferenced types and groups. The weight values of all elements and attributes are assigned on the basis of the weight values of their associated types. For example, the weight value for the global element A is equal to the weight value of its type and can be calculated by Eqs. (12.3) and (16.2),

$$\begin{aligned} we &= wcr_A \\ &= wcr_{baseType} \pm \left[\sum_{i=1}^N we_i E_i + \sum_{j=1}^M wa_j A_j + \sum_{t=1}^K weg_t EG_t + \sum_{s=1}^P wag_s AG_s \right] + NCR * R \\ &= 1 + \left[\sum_{i=1}^1 we_i E_i + \sum_{j=1}^1 wa_j A_j + \sum_{t=1}^1 weg_t EG_t + 0 \right] + 1 * R \\ &= 1 + [1 + 1 + 4] + R \\ &= 1 + [2 + 4] + R \\ &= 7 + R. \end{aligned}$$

The weight value for global element A is $7 + R$, where $R > 1$. Note that, the complex type A is implicitly derived by restriction from *anyType* of W3C XML Schema and its parent's weight value is considered to be 1. Further, the non-recursive region of the complex type A includes all its child elements and attributes except for the recursive child element S . The descendants of the complex type A , that are child of the element S are not included into non-recursive region, hence, their weight values are also not considered due to recursion. Instead, we count the number of child elements of the complex type A and multiply this number with the variable R having an integer value greater than 1. As a

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://exampleXMLSchema" xmlns:tns="http://exampleXMLSchema">
  <xs:complexType name="X">
    <xs:complexContent>
      <xs:extension base="tns:A">
        <xs:sequence>
          <xs:element name="D" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="c1" type="tns:unions"/>
        <xs:anyAttribute namespace="##other" processContents="lax"/>
        <xs:attribute name="b2">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:maxLength value="10"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:simpleType name="unions">
    <xs:union memberTypes="xs:anyType xs:integer xs:date"/>
  </xs:simpleType>
  <xs:element name="X" type="tns:X"/>
  <xs:group name="group">
    <xs:choice>
      <xs:element name="d" type="xs:dateTime"/>
      <xs:element name="e2" type="tns:unions"/>
    </xs:choice>
  </xs:group>
  <xs:complexType name="A">
    <xs:sequence>
      <xs:group ref="tns:group"/>
      <xs:any namespace="##any" processContents="lax"/>
      <xs:element name="S" type="tns:A" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="c2">
      <xs:simpleType>
        <xs:list itemType="xs:integer"/>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <xs:element name="A" type="tns:A"/>
</xs:schema>

```

Fig. 1. The schema document *exampleXMLSchema.xsd*.

result, the possible weight value for the complex type A can be $7 + R = 7 + 3 = 10$. Similarly, we evaluate the weight value for the element X as $14 + R$. Since the element X has complex type derived by extension from the complex type A having one recursive child element, the element X can also be considered recursive element. From Table 1 it can be observed that even the global elements d and $e2$ are simple typed elements their weight values hence, complexities are different (1 & 3 respectively) due to the difference in the internal architecture of their type definitions. For the similar reason, even the two global elements X and A have recursive complex type and each has only one recursive child element their weight values are different ($14 + R$ & $7 + R$) since the non-recursive region of the element X consist of more components than that of the element A .

Table 1. Weight values for different components.

Name	Notes	Weight		
		Symbol	Value	Eq. No
GLOBAL COMPONENTS				
X	A complex-typed element containing recursion	we	$14 + R$	12.3,16.2
A	A complex-typed element having containing recursion	we	$7 + R$	12.3,16.2
X	A complex type derived by extension with recursive parent	wcr	$14 + R$	16.2
A	A complex type having recursive child element, is derived by restriction from <code>anyType</code> by default	wcr	$7 + R$	16.2
<i>unions</i>	A simple type derived by union	ws	3	17.3
<i>group</i>	A global elements group	weg	4	15.1
LOCAL COMPONENTS				
D	A simple-typed element	we	1	12.1,17.1
$c1$	An attribute having simple type derived by union	wa	3	13.1,17.3
<i>anyAttribute</i>	An attribute declared by <code><anyAttribute></code> element of W3C XML Schema	wa	1	13.2
$b2$	An attribute having simple type derived by restriction	wa	2	13.1,17.2
S	A complex-typed recursive child element	we	$7+R$	12.3,16.2
$c2$	An attribute having simple type derived by list	wa	1	13.1,17.4
d	A simple-typed element	we	1	12.1,17.1
$e2$	A simple-typed element	we	3	12.1,17.1
<i>xs:any</i>	An element declared by <code><any></code> element of W3C XML Schema	we	1	12.4

These observations show that for a given schema document the metrics that are based on the count of the schema components and that neglect the internal architecture of the components does not give better indication about the complexity of that schema document than our proposed metric $C(XSD)$. That is, any two schema document having the same number of components may have different complexity values since the internal structure of these components may be different and this difference is better reflected by the proposed metric $C(XSD)$. In the example schema document, all of the complex and simple types are referenced as a type reference to either the elements or attributes and the element group is also referenced by the complex type A . Hence, while calculating $C(XSD)$ value for the example schema document we only consider the complexities of the two global elements X and A and their complexities reflected by their weight values are evaluated by calculating the weight values of their type definitions. The value of the $C(XSD)$ metric based on Eq. (11) is found $21 + 2R$ (and $R > 1$) for the example schema document shown in Fig. 1:

$$\begin{aligned}
C(XSD) &= \left[\sum_{i=1}^N we_i E_{gi} + \sum_{j=1}^M wa_j A_{gj} + \sum_{t=1}^K we_{gt} EG_{gt} + \sum_{s=1}^P wag_s AG_{gs} \right] \\
&\quad + \left[\sum_{r=1}^R wc_r cT_{gr} + \sum_{q=1}^Q ws_q sT_{gq} \right] \\
&= \sum_{i=1}^2 we_i E_{gi} + 0 + 0 + 0 + 0 \\
&= we_A + we_X \\
&= (7 + R) + (14 + R) \\
&= 21 + 2R.
\end{aligned}$$

4. VALIDATION OF C(XSD)

A newly proposed complexity measure is acceptable, only when its usefulness has been proved by a validation process. We evaluated and validated our metric both theoretically and empirically in sections 4.1 and 4.2 respectively.

4.1 Theoretical Validation

The necessity for practical evaluation and formal validation of any newly proposed metric is clear. For this purpose, in section 4.1.1, we examined our metric against a practical framework [18]. Measurement process is known to be critical in both science and engineering. In order to make the software discipline more and more mature we can use the tools provided by Measurement Theory (MT). As a consequence, a proposal of new software metric can be validated through the application of MT basics. In section 4.1.2, we will define the basics of MT [19] and look at it from measurement theoretical perspective.

4.1.1 Practical evaluation of C(XSD)

Practical success of any proposed metric depends on the establishment of (1) its validation, (2) understandability by its users and (3) tight link between the metric and the attribute that it is intended to measure. The establishment of (2) and (3) highly depends on the validation. Therefore, a new metric must be evaluated formally and practically for its validation. For practical evaluation of our metric, we followed the guidelines given by Kaner [18]. This approach for metric validation is more practical than the formal approach.

When we look C(XSD) from the perspective given in [18], it is an indirect metric. It is a function of numbers of components, which contributes to the measurement of software complexity. In the following paragraphs we evaluate our metric against the framework, which is based on the following points:

The purpose of the measure Two main purposes of our metric are to contribute to the judgment about schema quality and to provide a self-assessment and improvement for the developer.

Scope of usage of the measure The proposed metric can be categorized as a technical metric being applicable before after coding. Consequently, its scope of use is the software development group working specially for web services.

Identified Attribute to measure The attributes measured by our metric are the quality of the schema and the developer. More complex schema makes it less understandable and consequently less maintainable for future development effort.

Natural scale of the attribute The existence of natural scale for the attributes (but not the metrics) requires the development of a common, non-subjective view about them. We have no knowledge about the natural scale of attributes.

Natural variability of the attribute If an attribute involves human performance then we can talk about its variability. The reason behind it; although one can develop a sound approach to handle such attribute it may not be complete because of the existence of many other factors that affects the attribute's variability. The difficulty of making sound and complete empirical observations about the product results in no knowledge about the variability of the attribute.

Definition of metric The metric has been defined formally in section 2.

Measuring instrument to perform the measurement It uses the instrument of *counting* by either human or by machine. For automated counting purpose, one can easily develop a token generator and use the string matching algorithms.

Natural scale for the metric For the natural scale for our measure, we have to go through measurement theory. When we analyze our measure according to [19], we find that, it is on the ratio scale (see next section).

Relationship between the attribute to the metric value There is a direct relation between the quality of the schema and our metric $C(XSD)$. If the $C(XSD)$ value increases, it is clear that the schema quality will decrease since it implies inefficient use of memory and time. Note that $C(XSD)$ is not the unique indicator of schema quality and the same argument is true for the relation between the $C(XSD)$ value and the developer quality attribute.

4.1.2 $C(XSD)$ and measurement theory

The relation between measurement theory and evaluating criteria for software complexity measure is well established by several researchers. However, in general all of them suggest that the measure should fulfill some basic requirements based on measurement theory perspective. Amongst available validation criteria, the framework given by Briand *et al.* [19] is reported to be more practical. In this section, we adopt this framework since it also validates a given metric for various measurement concepts like size, length, complexity, cohesion and coupling.

Before applying our proposed measure against this framework, it seems appropriate to provide the basic definitions and the desirable properties for complexity measures given in the framework.

Definition Representation of Systems and Modules: A system S is represented as a pair $\langle E, R \rangle$, where E represents the set of elements of S , and R is a binary relation on E ($R \subseteq E \times E$) representing the relationships between S 's elements. Given a system $S = \langle E, R \rangle$, a system $m = \langle E_m, R_m \rangle$ is a module of S if and only if $E_m \subseteq E$, $R \subseteq E_m \times E_m$ and $R_m \subseteq R$.

For our proposed complexity measure, the entities are schema, *i.e.* E be a set of schema in S , the binary relation on schema is chosen to be greater than or equally complex.

Definition Complexity: The complexity of a system S is a function $\text{Complexity}(S)$ that is characterized by non-negativity, null value, symmetry, modular monotonic and disjoint module additivity properties.

In order to make it easier to follow the theoretical validation of our metric for the reader, the description of properties of Briand *et al.* [19] and corresponding evaluation of the proposed metric are given below:

Property Complexity 1 Nonnegative: The complexity of a system $S = \langle E, R \rangle$ is non-negative if $\text{Complexity}(S) \geq 0$.

Proof: Since our measure is obtained by the sum of non-negative number this property is satisfied.

Property Complexity 2 Null Value: The complexity of a system $S = \langle E, R \rangle$ is null if R is empty. This can be formulated as: $R = \emptyset \Rightarrow \text{Complexity}(S) = 0$.

Proof: Since no global elements and attributes, unreferenced global elements and attributes group, unreferenced global complex and simple type definitions/declaration are present in the XML schema document, the complexity value is trivially null and therefore this property is also satisfied by the proposed measure.

Property Complexity 3 Symmetry: The complexity of a system $S = \langle E, R \rangle$ does not depend on the convention chosen to represent the relationships between its elements: $(S = \langle E, R \rangle \text{ and } S^{-1} = \langle E, R^{-1} \rangle) \Rightarrow \text{Complexity}(S) = \text{Complexity}(S^{-1})$.

Proof: In the proposed measure, there is no effect on complexity value by changing its order or changing its representation because complexity value assigned to the schema document cannot depend on the order or way of representation. Therefore, this property is also satisfied by the proposed measure.

Property Complexity 4 Module Monotonicity: The complexity of a system $S = \langle E, R \rangle$ is no less than the sum of the complexities of any two of its modules with no relationships in common: $(S = \langle E, R \rangle \text{ and } m_1 = \langle E_{m_1}, R_{m_1} \rangle \text{ and } m_2 = \langle E_{m_2}, R_{m_2} \rangle \text{ and } m_1 \cup m_2 \subseteq S \text{ and } R_{m_1} \cap R_{m_2} = \emptyset) \Rightarrow \text{Complexity}(S) \geq \text{Complexity}(m_1) + \text{Complexity}(m_2)$.

Proof: The conditions $m_1 \subseteq S$, $m_2 \subseteq S$ and $E = E_{m_1} \cup E_{m_2}$, imply that no modification is made to the S when the system is partitioned into modules m_1 and m_2 .

In our case if any schema is partitioned into two schemas, the sum of the complexity values of its partitioned schema will never be greater than the complexity value of the joint schema. The complexity value of a joint schema is the sum of complexity value of

its components. Therefore, this property also holds by proposed complexity measure.

Property Complexity 5 Disjoint Module Additivity: The complexity of a system $S = \langle E, R \rangle$ composed of two disjoint modules m_1, m_2 , is equal to the sum of the complexities of the two modules: $(S = \langle E, R \rangle \text{ and } S = m_1 \cup m_2 \text{ and } m_1 \cap m_2 = \emptyset) \Rightarrow \text{Complexity}(S) = \text{Complexity}(m_1) + \text{Complexity}(m_2)$.

Proof: We have already proved this property in previous property. Our proposed metric satisfies this property because the interaction among schema does not have any effect on the resulting counted complexity values. If two independent schemas are combined into a single schema then the complexity values of individual schema will be combined. Therefore, this property is also proved by proposed measure.

As consequences of the above properties Complexities 1-5, it is shown that adding relationships between elements of a system does not decrease its complexity. Further our proposed complexity measure hold properties Complexities 1-5, therefore it is also applicable to the admissible transformation for the ratio scale. In other terms by fulfilling these properties, one may say that the proposed complexity measure is on the ratio scale, the most desirable property of complexity measure.

4.2 Empirical Validation

Empirical validation proves the practical utility of a new metric. For the validation of the proposed metric we analyzed 65 real example Schema files from the web. Note that some of these analyzed Schemas are extracted from WSDL [2] documents (the schema definition is given under `<types>` element of WSDL document). We assigned id numbers for each schema file for the sake of clarity and the references for these files are shown in Table 3. We applied our metric to these real examples and the corresponding values are given in Table 2. While evaluating the value for $C(XSD)$ we assumed that the value for the variable R is equal to the minimum integer number greater than 1 which is 2, since any complex type that is not explicitly derived by extension or restriction from any other types is implicitly derived from W3C XML Schema's *anyType* by restriction by default. We also calculated the values for the metrics [6, 8] that count the number of total complex type and global elements. The corresponding values of these metric and our newly proposed metric $C(XSD)$ are shown in the same table. The graphs depicted in Figs. 2 and 3 are drawn based on the data given in Table 2.

We have demonstrated in section 3, how to calculate the complexity of each components of the example Schema file shown in Fig. 1 and its overall complexity value measured by $C(XSD)$ metric. We have calculated the $C(XSD)$ value by adding complexity values of all *globally* defined/declared elements, attributes, unreferenced global element, attribute groups, unreferenced global complex and simple type definitions/declarations.

By considering the data collected in the Table 2 it can be observed from Figs. 2 and 3 that the $C(XSD)$ metric gives better indication about the complexity of a given Schema document than the metrics $\#CT$ and $\#E$ which are count based. As can easily be seen

Table 2. The analyzed schema files for the empirical validation of $C(XSD)$ metric. $\#CT$ is the total number of global and local complex types; $\#Eg$ is the number of global elements; $C(XSD)$ is the proposed metric; WR is the total weight of recursive and non-recursive regions of Schema document; $\#RE$ is the number of global recursive elements.

#ID	#CT	#Eg	$C(XSD)$	$R=2$	WR	$\#RE$
1	3	2	8			
2	4	4	7			
3	4	2	12			
4	5	5	12			
5	5	3	32			
6	6	21	81			
7	8	8	28			
8	8	8	22			
9	8	4	50			
10	8	12	20			
11	8	2	89			
12	8	8	18			
13	9	8	19			
14	10	6	48			
15	10	10	37			
16	11	8	27			
17	11	9	70			
18	11	8	18			
19	12	12	63			
20	13	11	163	163	147+8r	4
21	14	14	131	131	103+14r	7
22	14	14	30			
23	14	14	29			
24	14	9	23			
25	14	12	73			
26	15	10	61			
27	15	12	164			
28	16	8	54			
29	17	16	47			
30	17	16	44			
31	17	14	118			
32	18	8	52			
33	18	12	88			
34	18	12	70			
35	18	10	94			
36	18	14	179			
37	18	18	582	582	518+32*r	16
38	19	8	116			
39	20	15	93			
40	20	20	94			
41	21	16	55			
42	21	12	64			
43	21	6	50			
44	22	14	35			
45	22	14	60			
46	22	22	32			
47	23	22	54			
48	24	8	113			
49	24	18	47			
50	24	32	334			
51	25	12	532			
52	26	14	144			
53	27	11	275	275	271+2r	2
54	28	37	867	867	741+63r	18
55	30	18	108			
56	30	22	112			
57	30	24	162			
58	33	27	221			
59	34	34	83			
60	38	33	202			
61	39	29	405			
62	41	41	493	493	259+117r	31
63	43	76	840	840	166+337r	66
64	54	42	134			
65	69	8	349			

Table 3. Web references of analyzed schema files given in Table 2.

#ID	WEB LINK
1	http://www.thomas-bayer.com/axis2/services/BLZService?wsdl
2	http://www.elguille.info/NET/WebServices/HolaMundoWebS.asmx?WSDL
3	http://sdpws.strikeiron.com/SDPv1?WSDL
4	http://ws.strikeiron.com/InnerGears/CityStateByZip2?WSDL
5	http://sws-challenge.org/shipper/runner
6	http://www.retsinfo.dk/APIS
7	http://tripleasp.net/Services/ShowCode.asmx?WSDL
8	http://services.nirvanix.com/ws/Authentication.asmx?WSDL
9	http://service.ecocoma.com/shipping/fedex.asmx?WSDL
10	http://www.wubingstudy.com/WebService/Messages.asmx?WSDL
11	http://www.iperformonline.com/WebServices/employee/AddUpdateEmployee.asmx?WSDL
12	http://webservices.daelab.net/temperatureconversions/TemperatureConversions.wso?WSDL
13	http://rangiroa.essi.fr:8080/dotnet/evaluation-cours/EvaluationWS.asmx?WSDL
14	http://quisque.com/fr/chasses/blasons/search.asmx?WSDL
15	http://webservices.freshegg.com/resources/service1.asmx?WSDL
16	http://www.billyclark.com/DesktopModules/FotoVisionDNN/PhotoService.asmx?WSDL
17	http://gw1.aql.com/soap/sendsmsservice.php?wsdl
18	http://www.devhood.com/services/timelog/timelog-service.asmx?WSDL
19	http://trial.serviceobjects.com/ce/CurrencyExchange.asmx?WSDL
20	http://www.filigris.com/products/docflex_xml/xsddoc/examples/html/eclipse_uml2/index.html
21	http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rs.xsd
22	http://www.mathertel.de/AJAXEngine/S02_AJAXCoreSamples/CalcService.asmx?WSDL
23	http://services.test.musiccue.net/rapidcueapplication/SubmissionManager.asmx?WSDL
24	http://www.multispeak.org/interface/30j/10_OA_EA.asmx?WSDL
25	http://coolcampus.csse.monash.edu.au/MonashLibrary/LibraryMaps.asmx?WSDL
26	http://www.secureattachment.com/webservices/sadownload.asmx?WSDL
27	http://network.grandcentral.com/services/users/docliteral-v1
28	http://terraserver-usa.com/LandmarkService.asmx?WSDL
29	http://del.eterio.us/blog/editposts.asmx?WSDL
30	http://www.cts.com.pl/webservices/rt_info.asmx?WSDL
31	http://www.naf.no/loginservice/main.asmx?WSDL
32	http://www.geoservicios.com/V2.0/sgeo/sgeo.asmx?WSDL
33	http://itplaza.jeju.go.kr/rpt_ws/Rpt_Ws_FD.asmx?WSDL
34	http://demo.soapam.com/services/FedEpayDirectory/FedEpayDirectoryService
35	http://itplaza.jeju.go.kr/itplazaweatherservice/ItplazaWeatherService.asmx?WSDL
36	http://www.inphoto.cz/Service/PhotoServer.asmx?WSDL
37	http://www.filigris.com/products/docflex_xml/xsddoc/examples/html/eclipse_uml2/index.html
38	http://ws.strikeiron.com/BusinessDataAppend?WSDL
39	http://www.esendex.co.uk/secure/messenger/soap/InboxService.asmx?WSDL
40	http://www.sipeaa.it/wset/ServiceET.asmx?WSDL
41	http://www.oorsprong.org/websamples.arendsoog/ArendsoogbooksService.wso?WSDL
42	http://svc.exaphoto.com/eXaPhoto/CollectionServices.asmx?WSDL
43	http://ws.strikeiron.com/MidnightTraderFinancialNews?WSDL
44	http://www.simulation.fr/seq/SaintEtiQ.asmx?WSDL
45	http://pc218.cgk.affrc.go.jp/PMTTypeService/MainEntry.asmx?WSDL
46	http://metalmaker.net/metalmaker.asmx?WSDL
47	http://localhost/ogsa/services/GLCProcessService
48	http://ws.strikeiron.com/ReverseResidentialLookup?WSDL
49	http://acims9.acims.arizona.edu/PublicationDB/DEVSPubs.asmx?WSDL
50	http://ws.xwebservices.com/XWebBlog/V2/XWebBlog.wsdl
51	http://www.saiasecure.com/webservice/shipment/soap.asmx?WSDL
52	http://www.banguat.gob.gt/variables/ws/BDEF.asmx?WSDL
53	http://dev.w3.org/2006/xquery-test-suite/TestSuiteStagingArea/XQTSCatalog.xsd
54	http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rim.xsd
55	http://bible.sumerano.com/bible.asmx?WSDL
56	http://www.vbcentral.net/ShipService/Services.asmx?WSDL
57	http://www.xpyder.co.kr/XpyWebService/member.asmx
58	http://www.xignite.com/xNews.asmx?WSDL
59	http://hooch.cis.gsu.edu/bgates/MathStuff/Mathservice.asmx?WSDL
60	http://www.esendex.com/secure/messenger/soap/ContactService.asmx?WSDL
61	http://mlbs.net/nacgeoservices/geoservices.asmx?WSDL
62	http://www.codeplex.com/ajaxdoc/SourceControl/FileView.aspx?itemId=102696&changeSetId=7167
63	http://www.oasis-open.org/committees/regrep/documents/2.0/schema/query.xsd
64	http://www.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL
65	http://ws.strikeiron.com/GaleGroupBusinessInformation?WSDL

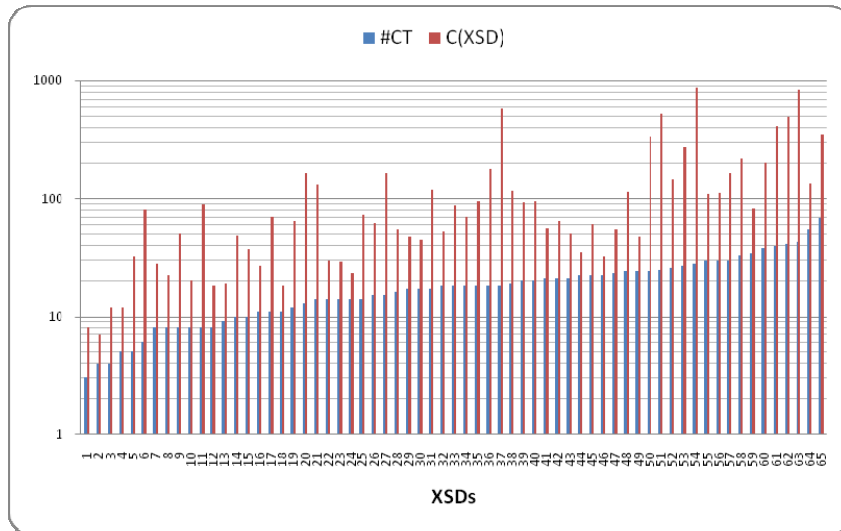


Fig. 2. #CT metric vs. $C(XSD)$ metric. The graph exploits the data given in Table 2. The data is ordered by #CT values.

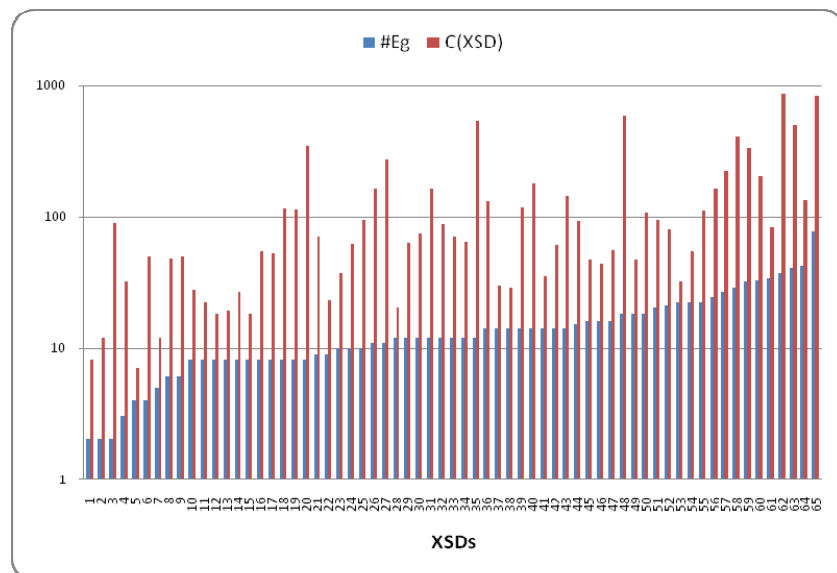


Fig. 3. #E_g metric vs. $C(XSD)$ metric. The graph exploits the data given in Table 2. The data is ordered by #E_g values.

from these graphs $C(XSD)$ metric evaluates different values for Schemas having equal #CT and #E values, thus can differentiate these XSDs in terms of their complexities. For a comparison according to the CT value the schema files having id number 65 should be the most complex one among the other schema files. However, according to the $C(XSD)$

value the most complex schema file having id number 54 is the most complex one even though if we assign an integer number 1 to the variable R representing the weight value of a recursive element. As we suggested earlier having greater number of recursive elements does not always result in for a given schema to have higher complexity. This suggestion can be supported by comparing the $C(XSD)$ values and the number of recursive elements ($\#RE$) of the Schema documents. Another suggestion stated earlier was that a given schema file that contains recursive elements may not always has higher complexity than a schema document that has no recursive elements. From Table 2 it can be observed that this suggestion can also be supported by making comparison between the values of $C(XSD)$ for the Schema files containing recursive elements and the values of $C(XSD)$ for the Schema files that does not contain recursion.

5. CONCLUDING REMARK AND FUTURE WORK

Flexible nature and ease of implementation of XML allows developer to create their own mark-ups to describe data, to define document types, to store, share and retrieve information and to transmit documents across web, thus, XML has been gaining a general acceptance as a standard not only for data representation but also for exchanging and retrieving information, since its development. In this aspect designing XML schemas play an important role in the software development process and needs to be quantified for ease of maintainability. For this purpose we have presented the complexity metric for the schema documents written in W3C XML Schema language. The proposed metric is evaluated on the basis of the internal complexities of major building components of XSDs and computed by using the provided formulas. Further, we demonstrated that the internal architecture of XSDs' building components affect the overall complexity of XSD. From this demonstration we can say that our complexity metric gives better indication than the metrics which measures the complexity of a given schema based on the counts of schema's each components. The proposed metric not only provides the complexity due to all components of the schema file but also due to the imported components from other external schema files, an important issue for complexity but not considered by the other metrics. In order to check the reliability of the proposed metric, $C(XSD)$ is theoretically evaluated and empirically validated. The theoretical and practical evaluations based on the information theory have shown that the proposed metric is on ratio scale and satisfies most of the parameters required by the measurement theory.

As a future work, we aimed to develop a new metric which measures structural complexity of XSD documents from their UML [20, 21], class diagrams. The applicability of the existing grammar metrics [22, 23] to the schema documents written in DTD, W3C XML Schema, RELAX is another future work since these schemas also be represented by tree grammars [24].

REFERENCES

1. <http://www.w3.org/TR/1998/REC-xml-19980210>.
2. E. Thomas, *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*, Prentice Hall Publishers, New Jersey, 2004.

3. <http://www.w3.org/TR/xmlschema-1/>.
4. ISO/IEC: Information Technology – Text and Office Systems – Regular Language, Description for XML (RELAX) – Part 1: RELAX Core, 2000.D TR 22250-1.
5. <http://www.xml.gr.jp/relax>.
6. A. McDowell, C. Schmidt, and K. Yue, “Analysis and metrics of XML schema,” in *Proceedings of International Conference on Software Engineering Research and Practice*, 2004, pp. 538-544.
7. M. H. Qureshi and M. H. Smadzadeh, “Determining the complexity of XML documents,” in *Proceedings of International Conference on Information Technology: Coding and Computing*, 2005, pp. 416-421.
8. R. Lammel, S. Kitsis, and D. Remy, “Analysis of XML schema usage,” in *Proceedings of XML*, 2005, pp. 1-35.
9. J. Visor, “Structure metrics for XML schema,” in *Proceedings of XML: Aplicações e Tecnologias Associadas*, 2006, pp. 1-10.
10. http://www.oreillynet.com/xml/blog/2006/05/metrics_for_xml_projects_1_ele.html.
11. M. Klettke, L. Schneider, and A. Heuer, “Metrics for XML document collections,” in *Proceedings of XMLDM Workshop*, 2002, pp. 162-176.
12. D. Basci and S. Misra, “Complexity metric for XML schema document,” in *Proceedings of the 5th International Workshop on SOA and Web Practices*, 2007, pp. 1-14.
13. <http://www.w3.org/TR/2001/PR-xmlschema-0-20010330/>.
14. E. van der Vlist, *XML Schema*, O’Reilly Publication, Sebastopol, 2002.
15. <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>.
16. <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>.
17. C. Binstock, D. Peterson, M. Smith, M. Wooding, C. Dix, and C. Galtenberg, *The XML Schema Complete Reference*, Addison Wesley Professional Publishers, Boston, 2002.
18. C. Kaner, “Software engineering metrics: What do they measure and how do we know?” in *Proceedings of the 10th International Software Metrics Symposium*, 2004, pp. 1-10.
19. L. C. Briand, S. Morasca, and V. R. Basily, “Property based software engineering measurement,” *IEEE Transactions on Software Engineering*, Vol. 22, 1996, pp. 68-86.
20. F. D. Salim¹, R. Price, M. Indrawan¹, and S. Krishnaswamy, “Graphical representation of XML Schema,” in *Proceedings of the 6th Asia-Pacific Web Conference*, LNCS 3007, 2004, pp. 234-45.
21. N. Routledge, L. Bird, and A. Goodchild, “UML and XML schema,” in *Proceedings of the 13th Australasian Database Technologies*, 2002, pp. 157-166.
22. T. Alve and J. Visser, “Metrication of SDF grammars,” Technical Report No. DI-PURE-05.05.01, Departamento de Informática, Universidade do Minho, 2005.
23. J. F. Power and B. A. Malloy, “A metrics suite for grammar-based software,” *Journal of Software Maintenance and Evolution*, Vol. 16, 2004, pp. 405-426.
24. <http://www.cs.ucla.edu/~dongwon/paper/>.



Dilek Basci completed her bachelor and master degree in Computer Engineering in 2004 and 2008 from Atilim University Ankara, Turkey. Presently, she is working as software specialist in BILGIGIS, a software company in Ankara. Her area of interest are object oriented design and programming, API design, database management, geographic information systems, internet programming, web technologies, GML, XML technologies and SOA. She is a motivated researcher and has produced some very good results in recent research papers.



Sanjay Misra is Assistant Professor in Department of Computer Engineering, Atilim University, Ankara, Turkey. Presently he is working in the area of Software Engineering, especially on software quality estimation through software metrics. His area of interests are software measurement, verification and validation techniques, object oriented technologies, XML, web services, and cognitive informatics. He published more than 40 research papers in these areas. Presently, he is chief editor of “International Journal of Computer Science and Software Technology” (IJCSST). He is also reviewer of several international journals and chaired several workshops and special sessions of international conferences.