MEASURING AND EVALUATING COMPLEXITY OF XML SCHEMA DOCUMENTS

A MASTER'S THESIS

in

Computer Engineering

Atilim University

by

DİLEK BAŞCİ

AUGUST 2008

MEASURING AND EVALUATING COMPLEXITY OF XML SCHEMA DOCUMENTS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

ATILIM UNIVERSITY

BY

DİLEK BAŞCİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

AUGUST 2008

Approval of the Graduate School of Natural and Applied Sciences, Atılım University.

_____

(Title and Name)

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

(Title and Name)

Head of Department

This is to certify that we have read the thesis "Thesis Name" submitted by "Candidates Name" and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____	_____
(Title and Name)	(Title and Name)
Co-Supervisor	Supervisor

Examining Committee Members

.....	_____
.....	_____
.....	_____
.....	_____
.....	_____

Date: (Thesis defence date)

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name:

Signature:

ABSTRACT

MEASURING AND EVALUATING COMPLEXITY
OF
SCHEMA DOCUMENTS

Başci, Dilek

M.S., Computer Engineering Department

Supervisor: Asst.Prof. Dr. Sanjay Misra

Agust 2008, 159 pages

The eXtensible Markup Language (XML) technologies have been promising to provide a common standard mechanism for interoperable integration of diverse IT processes and have been gaining extraordinary acceptance from the basic to the most complicated business and scientific processes. Further, many different domains, organizations and content providers have been publishing and exchanging information over the Internet by the usage of XML. Efficient implementation of XML in diverse 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 the ease of maintainability. The maintainability is one of the important factors that affect the quality of the software project and effective management of any type of software projects requiring modelling, measurement, and quantification. Further, software metrics should be used in order to improve the productivity and quality of software.

This thesis presents new metrics for XML schema documents written in W3C XML Schema Language and Document Type Definition (DTD) language. Additionally, as a related technology we also present a suite of metrics for the assessment of XML Web Services in terms of their maintainability.

Keywords: Software Metrics, XML, W3C XML Schema, DTD, XML schema Metrics, XML Web Services

ÖZ

XML ŞEMA DÖKÜMANLARININ KARMAŞIKLIĞININ ÖLÇÜMÜ VE DEĞERLENDİRMESİ

Başci, Dilek

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Yrd. Doç. Dr. Sanjay Misra

Ağustos 2008, 159 sayfa

Genişletilebilir İşaretleme Dili (XML) teknolojileri; farklı bilgi teknolojileri (IT)süreçlerinin işlevsel entegrasyonu için ortak bir standart mekanizma sağlanması konusunda umut vadetmekte ve en basitinden en kompleks ticari ve bilimsel işlemlere (süreçlere) kadar sıradışı bir kabul görmektedir. Dahası, XML kullanılarak birçok farklı alan, organizasyon ve içerik sağlayıcı internet üzerinden bilgi alışverişi yapabilmekte ve yayınlayabilmektedir. XML'in farklı alanlarda verimli kullanılabilmesi için iyi tasarlanmış XML şemaları gerekmektedir. Bu bakımdan, XML şemaları yazılım geliştirme süreçlerinde son derece önemli bir rol oynamakta ve kolay güncellenebilirlik için nicelendirilmelidir. Güncellenebilirlik, yazılım projelerinin etkileyen önemli faktörlerden biridir ve her hangi türden bir yazılım projesinin etkin yönetimi; modelleme, ölçümleme ve nicelendirmeyi gerektirir. Ayrıca, yazılımın kalitesini ve üretkenliği geliştirebilmek için yazılım metrikleri kullanılmalıdır.

Bu tez çalışmasında W3C XML Schema Language ve DTD ile yazılan XML şemaları için yeni metrikler sunulmaktadır. Bunun yanı sıra, XML teknolojileri ile bağlantılı olan XML Web Servisleri'nin güncellenebilirlik bakımdan değerlendilirmesi için bir dizi metrik sunulmaktadır.

Anahtar Kelimeler: Yazılım Metrikleri, XML, W3C XML Schema, DTD, XML şema metrikleri, XML Web Sevisleri

DEDICATION

To My Children

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor, Asst.Prof.Dr. Sanjay Misra, who guided me through the process of completing my thesis for my Master's degree. My special thanks go to Asst.Prf.Dr. Atila Bostan, who contributed his time, energy, and expertise to support my thesis.

I must also express my appreciation to my family especially to my husband Abdullah Başci, my children Sare and Furkan, my mother Ayşe Çaycı, my brother Şuayip Çaycı who have given me support and encouragement over the past several years, for their patience during the times when I should have been paying attention to them, but instead was absorbed in my studies. Last but not least, I wish extend my thanks to my friends Erhan Yıldırgan, Günhan Elif Evcil, and Şafak Özcan Doğan who has contributed even in a small way in completing the thesis. They never doubted my ability to achieve this important milestone. To all them many thanks.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

APO	Argument per Operation
ARS	Argument Repetition Scale
ARS	Argument Repetition Scale
C(XSD)	Complexity of XSD
DAC	Distinct Argument Count
DAC	Distinct Argument Count
DAG	Directed Acyclic Graph
DAR	Distinct Argument Ratio
DIT	Depth of Inheritance
DMC	Distnict Message Count
DMR	Distinct Message Ratio
DSERS	Distinct Structured Elements Repetition Scale
DSERS(DTD)	Distinct Structured Elements Repetition Scale of DTD
DTD	Document Type Definition
DW(wsdl)	Data Weight of WSDL
E (DTD)	Entropy of DTD
HTTP	Hyper Text Transport Protocol
IDE	Integrated Development Environment
LOC	Lines of Code
ME(wsdl)	Message Entropy of WSDL
MRS	Message Repetition Scale
NOC	Number of Children
OPS	Operations per Service
SE	Schema Entropy
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery, and Integration

W3C	Word Wide Web Consortium
WSDL	Web Service Description Language
XML	eXtensibel Markup Language
XSD	XML Schema Document

CHAPTER 1

INTRODUCTION

1.1 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. While information sharing among different parties connected to the Internet becomes widespread, the need for information to be transmitted from one party to another has resulted in the emergence of a standard mechanism for information exchange that can be easily implemented by different domains. The eXtensible Markup Language (XML) [1], 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 [2, 3, 60 and 61] based on XML technologies are emerging as the de-facto mechanism for exchanging structured information between organizations and applications. Due to flexible nature and ease of implementation, XML serves very well as a ubiquitous, platform-independent data representation and transport format and hence, has been easily adopted in diverse fields.

In XML context, the data representations are made by designing schema which can be written in different XML schema languages such as DTD [37, 62], W3C XML Schema [1], RELAX NG [65, 66]. W3C XML Schema and DTDs are the most favored schema languages for generating XML documents including the data. In order to capture the semantics of many different domains, organizations and content providers are beginning to publish and exchange information via the usage of XML and XML schema.

Efficient implementation of XML requires well designed XML schemas. In this point of view, design of XML schemas play an extremely important role in software development process and needs to be quantified for ease of maintainability. Neglecting schema 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 has to be performed by the application programs implying that the application developers have to write lengthy code. Using schema not only provides common understanding about exchanged data but also the ability of easy access methods for XML documents to be validated, thus, well designed schema provide a way for XML documents to be stored and retrieved efficiently and effectively. With the successful design of schema the developers can have the capability of increasing productivity, improving software reliability, minimizing development time, and decreasing time-to-market [31].

All above considerations related with XML documents imply that the schema documents need to be properly designed, so that they can be easily maintained in order for XML data to be effectively and properly used by diverse fields. Further, schema metrics must be developed –as for software products- to enable quantification of schema size, complexity, quality and the other properties. Although XML schema documents have been using in diverse fields of software industry and plays an important role in many software projects, up to present a few researches has been done in terms of quantification of XML schema documents through applying schema metrics. In this thesis we present a suite of metrics for the schema documents written in W3C XML Schema and DTD, since these are the most favored schema languages for generating XML documents. Additionaly, we also present a suite of metrics to evaluate and maintain the quality of the XML Web Service in terms of its maintainability since XML Web Services are one of the related technology. In the following paragraphs a brief introduction of existing metrics for XML documents is given.

1.2 A Brief Introduction of the Existing Metrics

To date many researchers has studied to develop metrics for software products. Although an extensive collection of online articles [5], publications [4, 6, 7, 8, 9, 10, 11], dealing with software measures for different type of software products available

only a few research has been done for the assessment of the quality of XML schema documents. The first attempt regarding XSDs was made in [16] which proposed eleven metrics for XSDs and two formulae that use the metrics to compute quality indices for XSDs and complexity indices for conforming XML documents. The metrics reported in that paper are mostly related with XSD components' counts such as number of elements, complex and simple types, annotations, type references, unbounded elements definitions/declarations. However they only provided one example XSD file to illustrate their metrics but, both empirical and theoretical validation of proposed metric were not presented. In [17] a comprehensive analysis was made to extract quantitative and qualitative information from actual 60 XML Schema documents and these documents were measured through systematic algorithms, on the basis of the intrinsic feature model of the XSD language and some metrics were presented for measuring XSD-agnostic Schema size considering number of all XML nodes, number of all element and attribute declaration, all types and model group definitions. In addition, the metrics LOC, McCabe were also revisited. Visser [18] also adopted some well known existing metrics developed for other software artifacts to XSDs to deal with structural complexity of XSDs but no theoretical validations of presented metrics have been done. Besides these papers many online articles about XSD metrics are also available on the web [38].

1.3 Contributions

In this thesis, we present a suite of metrics for the schema documents written in W3C XML Schema Language and DTD. In contrast to existing metrics the newly presented metrics for both XSD and DTD are verified by theoretical validations and except for DTD metrics all of XSD related metrics are also validated empirically. In addition, we also introduce metrics for measuring data complexity levels of XML Web Services since the usage of W3C XML Schema Language is mandated in these services [63]. These metrics are also verified by both theoretical and empirical validations.

1.4 Scope and Outline of the Thesis

The outline of this thesis is as follows.

Chapter 1: Discuss the need for developing XML schema metrics.

Chapter 2: Provides brief information about the emergence of XML, the need for schema documents and W3C XML Schema Language and the main features.

Chapter 3: Gives an overview on the different validation criteria for the software complexity metric, which includes the Weyuker's properties, and a framework for practical evaluation of software complexity metrics.

Chapter 4: Presents our metrics developed for schema documents written in W3C XML Schema Language.

Chapter 5: Presents a metric to evaluate the structural complexity of XML schema documents written in W3C Document Type Definition (DTD).

Chapter 6: Presents a suite of metrics for measuring data complexity levels of XML Web Services.

Chapter 7: Presents the conclusion for the thesis including future, and a summary of the main contributions of this thesis.

1.5 Typing Conventions

In this thesis, we shall strictly use "schema" with lower-case "s" to denote the general class of schema languages, and "Schema" with upper-case "S" to refer to the W3C XML Schema Language.

CHAPTER 2

e*X*TENSIBLE *M*ARKUP *L*ANGUAGE (*XML*)

2.1 Introduction

This chapter introduces brief information about the emergence of XML, the need for schema documents and W3C XML Schema Language. In section 2.2, we first discuss about the information exchange problem [22] which has led the invention of a new data representation languages or improvement of existing ones since the Internet was born. In section 2.3 the history of markup languages invented for information to be represented in a structured format is given. Section 2.4 provides an overview on eXtensible Markup Language (XML). The XML schema languages and XML schema documents are presented in section 2.5. In section 2.6 the importance of XML documents validation is emphasized. The last section summarizes W3C XML Schema Language and its building blocks.

2.2 Information Exchange Problem

The Internet technology is a primitive precursor of the Information Superhighway [20], a theoretical goal of computer communications to provide schools, libraries, businesses, and homes, universal access to quality information that will educate, inform, or entertain. The revolutionary benefit of the Internet since its invention [21] is connectivity. With the emergence of the World Wide Web the connectivity among people, diverse companies and business continues to proliferate at an astounding rate for accessing and exchanging a broad range of resources since the invention of the Internet. As the number of parties connected has been growing the information transferred among them has increased dramatically. With this growing connection the Internet accelerated the flow of information from device to device and server to server. By this flow of information the common problem of information exchange arose from several areas such as Web documents, e-commerce- database access,

knowledge sharing etc. and the achievement of common understanding about information has become an inevitable need for the industry. The underlying issue is that each party connected to the Internet to access a piece of information has a different perspective on what that information means.

As the Internet technologies has been evolving software solutions have required distributed computing where diverse applications run on different platforms needed to communicate and exchange application data. Toward the end of 1990s organizations have turned their attentions to Web applications. The last recent movement of the industry involves Web services as an extreme form of distributed computing.

The information exchange problem, in a distributed computing environment has become acute problem when increasing number of enterprises from different domains needed to connect to their partners, suppliers, and customers so that they could achieve efficiency in manufacturing, inventory, and distribution [22]. Such enterprises accumulated large volumes of data in various databases and encountered problems in gathering and delivering the data when they expanded people who have access to data. For these enterprises automated information processing was inevitable requirement and could be provided by enabling applications that run on diverse hardware and software platforms to interact with each other so that they could communicate directly, exchange and interpret the information and understand each other on business terms However, the same business terms such as product, order, and invoice could be differently defined in each partner's information systems. Even within one company, two divisions may have their own definitions of these business terms. As a consequence, parties from different domains needed a standardized format to describe the information to be transferred over the Internet and to facilitate the exchange and manipulation of data. In other words, information should have great deal of structure to be described and could be easily transferred so that parties either human or computers could easily interpret that information without losing its meaning. Without such structure the information accumulated during the chains of transfers could not be prevented from being misunderstood and confused.

2.3 Markup Languages for Structured Information

In the late 1960s IBM created GML (Generalized Markup Language) [23, 24] to address the needs of its internal editing and publishing systems. GML was mostly used to procedure reports, books, and other documents. At that time other solutions were also introduced within specific organizations and companies for structuring information but none of them were favored as GML. As GML more widely deployed its success has led IBM to the creation of SGML (Standardized Generalized Markup Language).

The significant power of SGML was due to the fact; it was designed not only to have GMLs flexibility and semantic richness but also to include concepts from information theory; inter-document link processing and a practical means to validate markup documents by ensuring the content conformed to a specific grammar. Having these features made SGML the most favorable markup language and fit for wide range organizations which needed to ensure consistency across vast repositories of documents. In 1986 SGML emerged as an ISO standard (ISO 8879).

Although SGML is extremely powerful due to its complexity a significant amount of software overhead is required for processing it. The Hypertext Markup Language (HTML) was born as a descendant of SGML for representing hypertext in the early of the Internet. HTML was never intended as a general means for structuring information exchanges since it only allows predefined a narrow set of metadata that only indicate how a document is presented visually in a web browser rather than indicating the meaning of information presented in the document . This limitation resulted in losing much of the flexibility offered by markup technology.

In the mid-1990s, the members of World Wide Web Consortium (W3C) [1] began to create a new subset of SGML for use on the Web. This new subset was supposed to be as a lighter tool reflecting the needs of the emerging web environment. The new markup language as a simplified descendant of SGML provided the flexibility and inherited best features of SGML, but could be processed faster. Six years later in 1996, the initial draft for this new "simplified SGML for Web"—the Extensible

Markup Language (XML) was presented by W3C and on 10 February 1998 the W3C published XML 1.0 [1] as an official recommendation.

2.4 eXtensible Markup Language

Instead of being a programming language XML is a meta markup language that offers three important benefits: extensibility, structure, and schema validation which were missing in HTML.

XML offers a very simple, straightforward, well-documented, portable data format. Since the documents represented by XML grammar are text documents that have the most portable and flexible format designed since the ASCII text file, they can easily be read by any tool being capable of reading a text file.

The data is described by surrounding some text markups called tags and readable by humans in XML representation of a document. In that representation the basic unit of data and markup is known as an element. All the markups used for describing the data must follow some rules such as how elements are delimited by tags, what a tag looks like, what names are acceptable for elements, where attributes are placed, and so forth. The standard data format offered by XML also allows describing the document's semantics but, contrary to HTML, it does not say anything about how the data should be displayed. This standard format is so simple, consistent and flexible enough that any program written in any language can accomodate it. Since the syntax of XML is well defined, parsers have quickly become available that can be easily incorporated into diverse programs. The existence of such parsers has also promoted XML usage as the language choice to transmit the data. By using XML, the documents and data represented in these documents can be transmitted from one platform to another and the system expecting these documents and data will be able to make sense out of it. These features enables XML to be customized for diverse domains such as web sites, electronic data interchange, remote procedure calls, voice mail systems etc. Hence, due to its abilities XML has become the de facto standard for transmitting data between distinct pieces of software and computers and particularly, represents new revolution for the Web by establishing a means of transmitting structured data.

The data transmitted as XML stream should be examined by the systems to know if the stream is correct or identifies errors, i.e. XML stream should be validated. By

validating XML documents against corresponding schema the receiving system can ensure that it gets what it expects. This is where XML schema language and XML schema documents come into play.

2.5 XML Schema Languages and XML Schema Documents

An *XML schema language* is a tool which is applied to a class of XML documents and implemented in the form of schema processors. There exist a variety of XML schema languages [80, 85] such as DTD[1, 37], W3C XML Schema[1], Relax NG[65, 66], SOX[82], Schematron [81] and comparison among them available in [83, 84, 85].

The structure of an XML document that conveys the data is described according to the rules of any XML schema language's grammars. This description is provided by an *XML schema documen*t which is a text document written in one of the schema languages. The XML 1.0 included a set of tools to define the structures of XML documents, which is known as Document Type Definition (DTDs) [1, 37]. As one of the most known XML schema languages, the definitions of the DTD's grammar, well-formedness of XML documents (documents that conform to the XML grammar) and valid XML documents (documents that conform to a DTD) were provided by W3C XML Recommendations. By using DTDs the structure of an XML instance document defined by the grammar rules such as which element and attribute structures are permitted, default values of attributes, reusable contents (entities) and some kinds of meta data information i.e. notations can easily be defined.

Although DTD worked very well, having lack of data typing and its vocabulary which is not XML vocabulary are perceived as major problems by many developers. Hence, the W3C sought to build a new schema language to describe XML documents. The new language XML Schema 1.0 [1] also referred as XML Schema Definition (XSD) introduced by W3C in 2001 and it addressed the need for more precision in describing XML documents structures, their contents and richer data types. The structures of XML documents were represented by using XML vocabulary rather than DTD's. The W3C's XML Schema Working Group [1] presented by two normative Recommendations, XML Schema *Part 1*: Structures [1] and XML Schema *Part 2*: Datatypes [1], along with a non-normative Recommendation, XML Schema *Part 0*: Primer [1].

At present the XML Schema Working Group as part of W3C XML Activity [1] is currently working towards the completion of XML Schema 1.1[1], which is intended to be mostly compatible with XML Schema 1.0 and to have approximately the same scope, consistent with the constraints on scope and compatibility [1].

2.6 Validity of XML Documents

The common feature of many schema languages is validation. A validation process as a most common use for schema documents is required for different reasons. The output data of an application may be required testing or an XML instance document may need to be validated with respect to a particular schema before the data conveyed by that instance document is imported into a legacy system, or exported from the external sources as is the case with Web Services [2, 3] and other XML communications.

In the validation process the schema processors take an XML document called instance document X, and a schema document S written in particular schema language and then checks if X is syntactically correct according to S. That is, if a given XML document X conveying data is conformed against its associated schema document S which is responsible for describing its structure then X is said to be valid otherwise diagnostic error messages are produced by the schema processors. Each XML document has to obey the well-formedness rule of XML grammar to be a valid document. However, this does not mean that all well-formed XML documents are valid documents. Validation is especially important in the distributed or *loosely coupled* scenarios where XML is utilized as a transfer format. By validating XML documents against the schema user can ensure that the contents of documents' conform to the expected set of rules, hence the code needed to process them is simplified. In this aspect XML schema specifies a contract between applications or between parts of a software application. This contract provides simplifying modularization, resource allocation, testing and deployment [31]. Modularizing the code is easier because the boundaries are readily identifiable (each boundary is an XML document). This in turn makes resource allocation easier: Individual developers receive specific tasks, each of which has well-defined inputs and outputs. Testing is also much easier: The developer generating XML ensures that the generated XML validates against the XML schema, and the developer receiving the

XML can easily create test XML documents in a common editor. Subsequently, integration testing is usually far more successful than with traditional data (such as passing objects and structures). Finally, even deployment is simpler: Versions of code that generate the XML can be deployed at different times than code that receives the XML—assuming that the schema is stable [31].

Due to machine-readability feature, XML schemas offer several advantages for documentation which may not require validation. Schemas can be used to document XML vocabularies since they provide a formal description of the vocabulary with a precision and conciseness that can be difficult to achieve in prose. The documents created from this formal description then can be readable by humans. For instance, Schema IDEs can automatically generate HTML documentation about XML data model, ensuring that the XML data model and documentation are in-synch. [32].

Although XML schemas are incredibly precise; they are also verbose and are not easy to learn. However once it is created the rest of software development process can be simplified which result in indirectly increasing productivity and decreasing time-to-market [31].

2.7 W3C XML Schema Language

W3C XML Schema can be distinguished from DTD by below main characteristics:

- Support of XML Namespaces [33]
- XML syntax
- Type derivation, similar to inheritance in object-oriented programming
- Support for definition of user-defined simple types, and definition of a set of built-in types
- Generalization of the concept of Identity Constraints
- Re-introduction of the all group, which had been omitted in DTDs

The above features make XML Schema more powerful and expressive, but at the same time harder to deal with than DTDs.

Although not part of the XML recommendation, XML Schema has become a core part in many of the W3C's XML-related recommendations. The most recent

generation of XML technologies, i.e., XSLT 2.0 [34], XPath 2.0 [36], and XQuery 1.0 [35] all build upon XML Schema.

2.7.1 Features

In this section we only give a brief list of the unique features of W3C XML Schema. For further explanations reader can refer to the XML Schema Primer [1].

XML Namespaces: XML Schema supports XML namespaces which is important during document validation. Namespaces provide a mechanism that prevents naming conflicts. Schemas usually define a target namespace, and the components defined by a Schema then have to be used by their qualified names. Given a target namespace which is mapped to the prefix *tns*, global components thus have to be referenced as follows:

```
<xsd:element name="paragraph" type="xsd:string"/>
<xsd:element ref="tns:paragraph" minOccurs="0"/>
```

Imports and Includes: XML Schema allows including of XML Schema documents having the same target namespace with including Schema, and importing of components from XML Schemas with different target namespaces. The *xsd:import* element may or may not specify the location of imported or included Schema via *schemaLocation* attribute:

```
<xsd:include schemaLocation="address.xsd"/>
<xsd:import namespace="http://www.w3.org/1999/xhtml"/>
```

Complex and Simple Types: Complex types describe the content model of elements, i.e., which child elements and attributes are permitted in which order. Simple types can restrict the literal content of elements that have simple type.

```
<xsd:complexType name="NameType">
<xsd:sequence>
<xsd:element name="First" type="xsd:string"/>
<xsd:element name="Last" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
```

Type Derivation: The user is allowed to define new types by type derivation mechanism of W3C XML Schema. Both complex and simple types can be used for

creating new types. Below Schema [1] illustrates type derivation mechanism. Complex types can be derived by restriction or extension. By using restriction method the content of a complex type can be narrowed and it can be extended by appending additional elements to the end of its model group via extension method. Simple types can only be restricted. User can derive a new simple types by restricting either other user defined simple types or built-in simple types. The derived simple types can have the set of permissible values and these values are narrowed via using *facets*. Below Schema example illustrates type derivation mechanism.

```
…

<complexType name="Address">
  <sequence>
   <element name="name"   type="string"/>
   <element name="street" type="string"/>
   <element name="city"   type="string"/>
  </sequence>
 </complexType>
 <complexType name="USAddress">
  <complexContent>
   <extension base="ipo:Address">
    <sequence>
     <element name="state" type="ipo:USState"/>
     <element name="zip"   type="positiveInteger"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
<complexType name="RestrictedPurchaseOrderType">
 <complexContent>
  <restriction base="ipo:PurchaseOrderType">
   <sequence>
    <element name="shipTo" type="ipo:Address"/>
    <element name="billTo" type="ipo:Address"/>
    <element ref="ipo:comment" minOccurs="1"/>
    <element name="items"  type="ipo:Items"/>
   </sequence>
  </restriction>
 </complexContent>
</complexType>
 <simpleType name="Postcode">
 <restriction base="string">
  <length value="7" fixed="true"/>
 </restriction>
</simpleType>
<simpleType name="UKPostcode">
 <restriction base="ipo:Postcode">
  <pattern value="[A-Z]{2}\d\s\d[A-Z]{2}"/>
```

```
 </restriction>
</simpleType>
….
```

In addition to derivation by restriction, simple types can be constructed by *list* or *union* mechanism. Union types and list types can then be further derived by restriction.

Wildcards: XML Schema provides wildcards *any* for elements and *anyAttribute* for attributes. A wildcard matches any element or attribute in an instance document, but the set of matching elements and attributes can be restricted to a set of namespaces. The validation process for matching items can be performed by Schema processor according to the values of *processContents* attribute, which can be *strict*, *lax* or *skip*.

```
<element name="htmlExample">
 <complexType>
  <sequence>
   <any namespace="http://www.w3.org/1999/xhtml"
      minOccurs="1" maxOccurs="unbounded"   processContents="skip"/>
  </sequence>
  <anyAttribute namespace="http://www.w3.org/1999/xhtml"/>
 </complexType>
</element>
```

Type Substitution: XML Schema provides a mechanism to force substitution for a particular element or type. Types can be substituted by derived types in instance documents and always must be indicated by the *xsi:type* attribute in the instance documents. Type substitution and derivation can be controlled in the Schema via the attribute *block* as given below Schema.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://cars.example.com/schema"
      xmlns:target="http://cars.example.com/schema">
 <complexType name="Vehicle" abstract="true"/>
 <complexType name="Car">
  <complexContent>
   <extension base="target:Vehicle"/>
  </complexContent>
 </complexType>
 <complexType name="Address" block="#all">
 <sequence>
  <element name="name"   type="string"/>
  <element name="street" type="string"/>
  <element name="city"   type="string"/>
 </sequence>
</complexType>
```

```
 <element name="transport" type="target:Vehicle"/></schema>
```

And a valid instance document:

```
<transport xmlns="http://cars.example.com/schema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="Car"/>
```

Substitution Groups: XML Schema's substitution groups allow elements to be substituted for other elements. Elements can be assigned to a special group of elements that are said to be substitutable for a particular named element called the *head* element. Both the *head* and the elements within a substitution group have to be declared as global. The types of elements within a substitution group must have the same type as the *head* element or can be derived from the type of the group *head.* In an instance document, occurrences of the head element are allowed to be replaced by elements from the substitution group. In the below example Schema the element shipComment can be replaced with comment element in the instance document.

```
<element name="shipComment" type="string"
      substitutionGroup="ipo:comment"/>
```

Identity Constraints: XML Schema expands the concept of ID/IDREF of DTDs. It offers three kinds of constraints that are defined within element declaration: *unique*, *key*, and *keyref* as can be seen from below Schema.

```
<element name="purchaseReport">
  <complexType>
   <sequence>
    <element name="regions" type="r:RegionsType"/>
    <element name="parts" type="r:PartsType"/>
   </sequence>
   <attribute name="period"     type="duration"/>
   <attribute name="periodEnding" type="date"/>
  </complexType>
  <unique name="dummy1">
   <selector xpath="r:regions/r:zip"/>
   <field xpath="@code"/>
  </unique>
  <key name="pNumKey">
   <selector xpath="r:parts/r:part"/>
   <field xpath="@number"/>
  </key>
  <keyref name="dummy2" refer="r:pNumKey">
   <selector xpath="r:regions/r:zip/r:part"/>
   <field xpath="@number"/>
  </keyref></element>
```

The defined constraint act on element or attribute declarations. A *selector* defines the set of nodes among which the constraint should be asserted, and one or more *field*s determine the constrained values of those nodes.

All Groups: As a compositor *all* groups must be used alone within model group definition of a complex type and particles in *all* groups may occur at most once.

Element Declaration Consistent Rule (EDC): The element declaration consistent rule prohibits elements to have the same name with different types, within a given model group, i.e. each element in a model group must be unique.

Unique Particle Attribution (UPA): The unique particle attribution rule is a mechanism to prevent ambiguity. UPA [1, 39] says that in order for a given content model to be legal, there can be no instance that would have an element matching more than one *particle*. Consider the following content model which violets UPA:

```
<sequence>
        <element ref="a" minOccurs="0"/>
        <element ref="a" minOccurs="0"/>
  </sequence>
```

When processor tries to validate an instance <a/> it finds two matching items in the Schema and since both elements are declared as optional (through *minOccurs="0"*) it cannot "uniquely attribute" the element <a/> to a particle. UPA is violated.

CHAPTER 3

EVALUATION AND VALIDATION CRITERIA FOR SOFTWARE METRICS

"***What is not measurable make measurable***", Galileo Galilei (1564 - 1642)

3.1 Introduction

The newly proposed metric is acceptable only when its usefulness has been proven by a validation process. The purpose of the validation is also to prove the usefulness of an attribute which is measured by the proposed metric. For theoretical validation several researchers' proposed different criteria [4, 8, 10, 11, 41, 69, 73] against which the proposed software metric should adhere. Amongst the available theoretical evaluation criteria, evaluation through Weyuker's properties [73] is mostly used by the developers of new complexity measure. Weyuker's properties play an important role in evaluating software complexity measures. These properties are used to evaluate the robustness of a measure and in turn lead to the definition of good notions of software complexity. Although these properties are under several criticisms; still they are used for evaluation purpose and topic of research. Nevertheless, we used these properties for evaluation of our proposed metrics. Short descriptions of these properties are given in section 3.2. Another way of theoretical validation of software metrics named the measurement theory is revisited in section 3.3. An alternative approach to metric validation, which is more practical than the formal approach, is given by Kaner [69]. This practical framework is based on "how to recognize and describe the attribute in the empirical observation domain in order to relate its values to the proposed metric". We gave the short description of this framework in section 3.4.

3.2 Weyuker's properties

Elaine Weyuker [73] has brought together nine properties that software metrics should posses certain properties to increase their level of confidence and usefulness. These properties also evaluate the weaknesses of a measure in a concrete way and in turn lead to the definition of really good notions of software complexity. With the help of these properties one can determine the most suitable measure among the different available complexity measures. Although these properties are very much criticized by several researchers, they are still used as a standard and formal list of requirements against which to evaluate newly proposed software metrics. Weyuker's properties include notions of non-coarseness, non-uniqueness, monotonicity, interaction, and permutation. It is worthy to say that most of the well known metrics do, in fact, not satisfy one or more of these desired features according to Weyuker's properties. In the following paragraph the Weyuker properties are given in brief.

Property1. $(\exists P)\,(\exists Q)(|P| \neq |Q|)$.Where P and Q are program body.
This property states that a measure should not rank all programs as equally complex. It is concluded that any complexity measure, which rank all program, as equally complex cannot take as good complexity measure. Because if any measure rank all program as equally complex, there is no meaning of the complexity measure [89], which cannot differentiate between all possible programs. Therefore this property is essential property for any complexity measure, so complexity measure should satisfy this property.

Property 2: Let c be a non-negative number then there are only finitely many programs of complexity c.

This property states that there are only a finite number of programs of the same complexity. Most of the complexity measure should satisfy this property; because there are only finite number of programs of the same complexity. Weyuker's herself says that a measure is not sensitive enough if it divides all programs into just complexity classes. Property 2 is an attempt to formalize this intuition. For any complexity measure, one may assume some largest possible number that can be taken as an upper bound. This upper bound may be function of the particular machine used, and will be assumed to exist.

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

This property states that even if there exists different programs, the complexity of these programs may be the same i.e. there are multiple programs of the same complexity.

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

This property states that even though two programs compute the same function, the program complexity is determined by implementation details.

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

This property states that measurement for the combination of two program entities can never be less than the measurement made for either of the components program entity.

Property 6 a: $(\exists P)\ (\exists Q)\ (\exists R)(|P| = |Q|\ \&\ |P; R| \neq |Q; R|)$.

Property 6 b: $(\exists P)\ (\exists Q)\ (\exists R)(|P| = |Q|\ \&\ |R; P| \neq |R; Q|)$.

This two properties state that interaction between P and R can be different than interaction between Q and R resulting in different complexity values for (P; R) and (Q; R), where complexities of P and Q are the same. It is also an important property and worth considering for evaluation purpose.

Property 7: There are program bodies P and Q such that Q is formed by permuting the order of the statements of P, and $(|P| \neq |Q|)$.

Property 8: If P is renaming of Q, then $|P| = |Q|$.

This property states that uniformly changing variable name should not affect a program's complexity.

Property 9: $(\exists P)\ (\exists Q)\ (|P| + |Q|) .< (|P; Q|)$.

This property allows for the possibility that as program grows from its component program bodies, additional complexity is introduced i.e. a combined program may be more complex than its constituent parts.

3.3 Measurement Theory

Another way of theoretical validation of software metrics is through measurement theory. The relation between measurement theory and evaluating criteria for software

complexity measure is well established by several researchers. Accordingly, new software complexity measure should also be satisfied by the measurement theory criteria. On the other hand, measurement theory has a major problem in defining empirical observations on software entities in terms of their measured quantities [89]; therefore the existence of quantification is one of the major problems for validation. Further, IEEE 1061 standard tries to solve this problem by suggesting the use of a direct metric which does not depend on a measure of any other attribute and assumed to be valid by itself. However, Kaner and Bond [69] found this approach a weak and risk-prone substitute for a casual model. In their work, they also questioned the usage of direct metrics for the quantification problem of practical evaluation through the user-dependent, subjective and not being single but multidimensional function characteristics of them. The authors have shown that the characteristic of being multidimensional function contradicts with the direct metrics. As a consequence, we will not evaluate our proposed metrics against measurement theory and IEEE standards.

3.4 A Practical Framework for Evaluating Metrics

The practical success of any proposed metric depends on the establishment of (1) its validation, (2) understandability by its users and (3) a tight link between the metric and the attribute that it is intended to measure. Therefore, a new metric must be evaluated practically and formally for its validation.

However, in practical approach, the main concern is "how to recognize and describe the attribute in the empirical observation domain in order to relate its values to the proposed metric". Fenton [86] suggested that a metric is valid if it can be shown that it gives a proper numerical characterization of some attributes and suggest to follow representational measurement theory. On the other hand, the representational measurement theory does not care about the practical difficulty of making empirical observations on the attribute and their identification [87]. Note that, the *existence* of scale is still one of the major problems of the field. The IEEE 1061 standard [88] tries to solve the quantification problem by suggesting the use of *direct metric*, which does not depend on a measure of any other attribute and assumed to be valid by itself. Other metrics are validated in terms of direct metrics. An alternative approach to metric validation, which is more practical than the formal approach, is given by

Kaner [69]. We follow his guidelines for practical evaluation of our metrics. According to a practical framework for evaluating software metrics proposed by Kaner the following questions should be answered:

What is the purpose of this measure?: The purpose of the measure include evaluating project status, evaluating staff performance, facilitating private self-assessments and improvement, informing others about characteristics(such as development status or behavior of product),informing external authorities about the characteristics of the product.

What is the scope of this measure? : The scope of the measure should be clearly defined. Examples are: a single method from one person, one project done by one workgroup, a year's work from the workgroup or the entire company's output. As the scope broadens, more confounding variables can come into play, potentially impacting the metric. A metric works well in a specific region, but fails globally.

What attribute are we trying to measure? : It should be clearly defined that what attribute we are trying to measure. Either it is quality of product, effectiveness of testing, thoroughness of testing, effectiveness of the tester, skill or diligence of the programmer, reliability of the product etc.

What is the natural scale of the attribute we are trying to measure? : One should know the natural scale of the attribute to be measured. It is also possible that there is no knowledge of the natural scale of the attribute, e.g. what type of scale makes sense for programmer skill or thoroughness of testing or size of a program?

What is the natural variability of the attribute? : One should know the knowledge of the variability of the attribute, because there may be variability in anything that involves human performance.

What is the relationship of the attribute to the metric value? : The relationship between attribute and metric should be known. How much the attribute affect the metric.

What is the definition of this measure and what measuring instrument do we use to perform the measurement? : Define the function that assigns a value to the attribute. In addition, the way or the method or instrument, by which it is measured. Example, for the attribute length, one may use ruler (the instrument) and read the number from it.

What is the natural scale for this metric? : The scale of the metric should be decided. It is either interval or ratio scales. The scale of the metric may be different from the scale of the attribute.

What are the natural and foreseeable side effects of using this instrument? : In order to improve the measured result; if we change our circumstances or behavior, what impact are we going to have on the attribute?

These above questions point the fundamental concept of a new proposed complexity measure. These points show that the developer has a clear concept about their proposal and he/she knows what is going to measured and for what i.e. if the proposed metrics are meaningful or not.

CHAPTER 4

W3C XML SCHEMA METRICS

4.1 Introduction

It is well known that the maintainability is one of the important factors that affect the quality of any kind of software projects. As stated in chapter 1 XML Schema play an important role in software development projects and need to be properly designed, so that it can be easily maintained in order for XML data to be effectively and properly used by diverse fields.

In this chapter we present three complexity measures for the assessment of a Schema quality in terms of its maintainability. The first metric Schema Complexity *C(XSD)* metric [67] for W3C XML Schema is introduced in section 4.2. Contrary to the other metrics, *C(XSD)* calculates internal complexities of each Schema component. The second metric Schema Entropy *(SE)* and the third measure Distinct Structured Element Repetition Scale *(DSERS)* are presented in section 4.3 and 4.4 respectively. Both *SE* and *DSERS* metrics are based on entropy concept; since entropy concept is successfully adopted as a measure of complexity in most programming languages and not yet been extended and validated for the assessment of XML Schema documents. Therefore we adopted these metrics for measuring Schema complexities. These metrics exploit a directed graph representation of Schema document and consider the complexity of Schema due to its similar structured elements and the occurrences of these elements. The empirical and theoretical validations of all three metrics are given in the related sections.

4.2 Schema Complexity Metric *C(XSD)*

Schema complexity metric is based on the internal complexities of each Schema component and is intended to measure physiological complexity of a given Schema document. By physiological complexity we refer the effort required to comprehend

the Schema document. As the effort needed for understanding the Schema increases so does the Schema complexity. Thus *C(XSD)* metric is designed to capture how internal complexity of each Schema component affects understandability of overall Schema and its maintainability.

4.2.1 Motivation

Numbers of researchers [16, 17, and 18] have introduced several techniques to measure the complexity of the Schema documents. However, most of them concentrate to count the number of Schema components for calculating the complexity of XSD. On the other hand, we believe that the complexity of a given Schema document is highly influenced by the complexities of its components' internal architectures, that is, each component of a Schema contributes their complexity values on the basis of their design architectures to the Schema document's complexity.

Consider the following two example Schema documents given in listing 4.1 and listing 4.2 respectively. Both Schema documents have three global complex type definitions and five elements in total. According to the *#CT* metric both Schema documents are on equal complexity level. However we believe that this is not case since content models of complex type definitions can be constructed in different ways. When a developer navigates the first Schema document to understand the structure of global element x2 he/she first refers to its type definition which is C2.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="example1" xmlns:tns="example1">
<xsd:element name="X2" type="tns:C2"/>
  <xsd:complexType name="C1"/>
 <xsd:complexType name="C3"/>
  <xsd:complexType name="C2">
        <xsd:sequence>
          <xsd:element name="a1" type="xsd:string"/>
          <xsd:element name="a2" type="tns:C1"/>
          <xsd:element name="a3" type="xsd:string"/>
          <xsd:element name="a4" type=" tns:C3"/>
          <xsd:element name="a5" type=" xsd:string"/>
        </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Listing 4.1 Example Schema Document

While investigating the content model of C2 he/she has to exit out of the C2 block and turn back the definitions of complex types C1 and C3 to understand the structures of local elements a2 and a4. Since both C1 and C3 have empty content they can easily be comprehended without exploring any other element's type structure, thus by making less effort what the developer sees is that the global element x2 has five child elements.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="example2" xmlns:tns="example2">
<xsd:element name="x2" type="tns:X2"/>
  <xsd:complexType name="X1">
    <xsd:sequence>
      <xsd:element name="a1" type="xsd:string"/>
      <xsd:element name="a2" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="X3">
    <xsd:sequence>
      <xsd:element name="a5" type="tns:X1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="X2">
    <xsd:complexContent>
      <xsd:extension base="tns:X1">
        <xsd:sequence>
          <xsd:element name="a3" type="tns:X1"/>
          <xsd:element name="a4" type="xsd:X3"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

Listing 4.2 Example Schema Document

On the other hand, when the developer tries to understand the structure of global element x2 in second Schema (see listing 4.2) he/she should first investigate its type definition which is X2. Then, to understand the content of complex type X2 the developer should also visit the complex type definition X1, since X2 is derived by extension from X1 and the local element a3 has a type reference to X1. Moreover, the complex type X3 should also be navigated to comprehend the structure of local element a4 declared within X2.

Since in the second Schema there is no complex type definition having empty content the developer has to make more effort to know the structures of each element declared within these three complex type definitions. Thus, he/she has to make more effort to comprehend the structure of the global element `x2` declared in the second Schema than the element `x2` declared in the first Schema. Therefore it is our opinion that both Schemas cannot have equal complexity even they have the same value for *#CT*.

Based on the definition of *C(XSD)* (see following subsection) these Schemas will have different complexities since it considers complexities of component's internal architecture which is not considered by other researchers.

4.2.2 Definition of *C(XSD)*

What the *C (XSD)*[67] metric assumes is that each component of a Schema may require different effort to be comprehended and hence should be assigned a weight value reflecting the effort needed to understand the internal structures of each component defined in XSD.

This weight value is called *complexity degree*. Thus, to evaluate a single complexity value measured by *C(XSD)*metric for a Schema document each of its component's weight values i.e. complexity degrees are summed up.

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 [1]. The Schema document may not necessarily validate any XML document and can be designed as a library document or for documentation purposes. Based on its design style [39] 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. A 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 [1, 31, and 39] 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 and built-in simple type and complex type definitions.

d) The complexity due to elements definitions/declarations that contain recursion

e) The complexity due to components that are included or imported from external Schema files.

Accordingly the total complexity of the XSD can be defined by the following formula:

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

where *C(XSD)* is the overall complexity value of the Schema document; $C(V_g)$ is the total complexity values of all global elements and attributes; $C(G_g)$ is the total complexity values of *unreferenced* global elements and attributes group, and $C(T_g)$ is the total complexity values of *unreferenced* global complex and 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. However, it is not common that a given Schema document includes unreferenced components.

The complexity of the Schema document is also highly affected by the elements that contain recursion. An XML Schema is said to be *recursive* when the type definition of an element in it allows for elements of the same name and type to appear in their own definition [40]. 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 elements is more complex than any other element having less recursive child elements or has only non-recursive elements or attributes. It is also arguable that as the number of recursive elements increases in the Schema document so does the effort required comprehending the Schema document.

In our opinion, it is possible that the effort required to understand the Schema including recursive elements may be less than that of Schema which 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 participant 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$_g$), C(A$_g$)* are complexities of global elements and attributes definitions/declarations respectively and :

$$C(E_g)=\sum_{i=1}^{N} we_i \, E_{g_i} \quad (3)$$

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

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_j} 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:

$$C(G_g) = C(EG_g) + C(AG_g) \quad (5)$$

where *C(EG$_g$), C(AG$_g$)* are the complexity of global elements and attributes group definition/declaration respectively and are defined as:

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

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

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:

$$C(T_g)=C(cT_g) + C(sT_g) \quad (8)$$

where *C(cT$_g$), C(sT$_g$)* are complexity of global complex and type definition respectively and are defined as:

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

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

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

$$C(XSD) = \left[\sum_{i=1}^{N} we_i \, E_{g_i} + \sum_{j=1}^{M} wa_j \, A_{g_j}\right] + \left[\sum_{t=1}^{K} weg_t EG_{g_t} + \sum_{s=1}^{P} wag_s AG_{g_s}\right] + \left[\sum_{r=1}^{R} wc_r cT_{g_r} + \sum_{q=1}^{Q} ws_q sT_{g_q}\right] \quad 11.$$

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 [1, 31, and 39]. As a result, while the weight value of an 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* [1, 31, and 39] with different number of occurrences 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 its content model 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 have simplest data type structure used in the Schema document. In a Schema document an element that does not explicitly specify its data type implicitly specifies *anyType* [1, 31, and 39] as its data structure type. The content of an element in an XML instance whose data 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* element of W3C XML Schema Language is 1. The <*any*> [1, 31, and 39] wildcard provides a mechanism for defining elements. 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 in the Schema 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*>[1, 31, and 39] attribute wildcard of W3C XML Schema Language is 1 since <*anyAttribute*> element is analogous to the <*any*> element of W3C XML Schema. Another point that needs to be paid 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 [1, 31, and 39] while evaluating complexity value of the current Schema document. Based on these assumptions, weight values for each Schema component can be calculated as follows:

Element's weight value *we* is:

we=

$$\begin{cases} \boldsymbol{ws}, if\ element\ has\ simple\ type & 12.1 \\ \boldsymbol{wc}, if\ element\ has\ complex\ type & 12.2 \\ \boldsymbol{wcr}, if\ element\ is\ global\ and\ has\ complex\ type\ having\ recursion & 12.3 \\ \mathbf{1},\ if\ element\ is\ declared\ by\ using\ <any>\ element & 12.4 \\ \mathbf{1},\ if\ element\ is\ declared\ by\ using\ \boldsymbol{anyType} & 12.5 \\ R,\ if\ element\ has\ recursion\ and\ is\ local & 12.6 \end{cases}$$

While calculating the weight value of an element we consider whether that element contains recursion or not. If an element does not contain any recursive child elements then 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 a recursive child element we further 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* given by equation 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 of any recursive child element may go to 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 :*

wa=

$$\begin{cases} \boldsymbol{ws}, since\ attributes\ can\ only\ have\ simple-type & 13.1 \\ \mathbf{1}, if\ attribute\ is\ declared\ by < anyAttribute > element & 13.2 \end{cases}$$

Weight values of elements group *weg* can be calculated based on its definition mechanism:

***weg*=**

$$\begin{cases} \sum_{i=1}^{N} we_i E_i,\ if\ not\ redefined & 14.1 \\ weg_{baseGroup} + \sum_{i=1}^{N} we_i E_i\ ,\ if redefined\ by\ extension & 14.2 \\ weg_{baseGroup} - \sum_{i=1}^{N} we_i E_i,\ if\ redefined\ by\ restriction & 14.3 \end{cases}$$

where $weg_{baseGroup}$ 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 equation 15.1 represents the number of elements if the group is not redefined. In equation 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 equation 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 [1, 31, and 39].

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

***wag*=**

$$\begin{cases} \sum_{i=1}^{N} wa_i A_i\ , \text{if not redefined} & 15.1 \\ wag_{baseGroup} + \sum_{i=1}^{N} wa_i A_i\ , \text{if redefined by extension} & 15.2 \\ wag_{baseGroup} - \sum_{i=1}^{N} wa_i A_i, \text{if redefined by restriction} & 15.3 \end{cases}$$

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

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

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

, 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, *wcr*, can be calculated in a similar way that *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.

$$\boldsymbol{wcr} = \mathrm{wcr}_{\mathrm{baseType}} \pm \left[\sum_{i=1}^{N} we_i\, E_i \;+\; \sum_{j=1}^{M} wa_j\, A_j \;+\; \sum_{t=1}^{K} weg_t EG_t \;+\; \sum_{s=1}^{P} wag_s AG_s\right] + \mathrm{NRC}*\mathrm{R} \qquad 16.2$$

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 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,* which contains recursive child (branches) the weight values of its recursive descendents are not considered due to the recursion.

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

ws=

$$\begin{cases} 1, if\ it\ is\ built\ in\ simple\ type & 17.1 \\ r+1, r\ is\ the\ number\ of\ restriciton, if\ it\ is\ derived\ by\ restriction & 17.2 \\ u|u=\sum_{i=1}^{P} w_i M_i\ ,\ if\ it\ is\ derived\ by\ union & 17.3 \\ l|l\ is\ the\ weight\ walue\ of\ item\ type,\ if\ it\ is\ derived\ by\ list & 17.4 \end{cases}$$

In equation *17.3* the capital *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 [1, 31, and 39].

4.2.3 Illustration of the *C(XSD)*

We have demonstrated *C(XSD)* metric by taking an example exampleXMLSchema.xsd document given in listing 4.3 . We calculated the weight values of each component declared/defined inside the given example Schema, and these values are shown in Table 4.1.

Example 4.1:While calculating the complexity value for the example Schema document given in listing 4.3 we sum all the weight values of all global elements, attributes and 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 instance, the weight value for the global element A is equal to the weight value of its type and can be calculated by 12.3 and 16.2:

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

and, R>1.

```
<?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>
```

Listing 4.3 The Schema document exampleXMLSchema.xsd.

Table 4.1 Weight values for different components of exampleXMLSchema.xsd.

Name	***Notes***	***Weight***		
		Symbol	***Value***	***Equation 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 anyType by default*	*wcr*	*7+R*	*16.2*
unions	*A simple type derived by union*	*ws*	*3*	*17.3*
group	*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*
anyAttribute	*An attribute declared by <anyAttribute> 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*
xs:any	*An element declared by <any> element of W3C XML Schema*	*we*	*1*	*12.4*

Note that, the complex type A is implicitly derived by restriction from base type *anyType* of W3C XML Schema and its base type'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 children 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 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 4.1 it can be observed that even the global elements d and e2 are simple typed elements their weight values reflecting their complexities are different (1 and 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 and 7+R respectively). This is due to the fact; the non-recursive region of the element X consists of more components than that of the element A.

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 documents having the same number of components may have different complexity levels since the internal structure of these components may be different and this difference is better reflected by the proposed metric *C(XSD)*. In exampleXMLSchema.xsd, 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 exampleXMLSchema.xsd we only consider the complexities of the two global elements *X* and *A*..

Example 4.2: The value of the *C(XSD)* metric through equation 11 is found $21+2R$ (and R>1) for exampleXMLSchema.xsd :

$$C(XSD) = \left[\sum_{i=1}^{N} we_i \, E_{g_i} + \sum_{j=1}^{M} wa_j \, A_{g_j}\right] + \left[\sum_{t=1}^{K} weg_t EG_{g_t} + \sum_{s=1}^{P} wag_s AG_{g_s}\right] + \left[\sum_{r=1}^{R} wc_r cT_{g_r} + \sum_{q=1}^{Q} ws_q sT_{g_q}\right]$$

$$= \sum_{i=1}^{2} we_i \, E_{g_i} + 0 + 0 + 0 + 0$$

$$= we_A + we_X$$

$$= wcr_A + wcr_X$$

$$= (7+R) + (14+R)$$

$$= 21 + 2R$$

According to the definition of *C(XSD)* the complexity values of example Schema documents shown in listing 4.1 and listing 4.2 are calculated as 6 and 11 respectively. Although both Schemas have the same *#CT* values and equal number of elements measured by the metric presented in [16, 17 and 38] their *C(XSD)* values are different. From this observation we can say that the *C(XSD)* metric can better reflect the complexities of XSDs and, therefore it is a useful metric to rank and differentiate XSDs.

4.2.4 Validation of *C(XSD)*

A newly proposed complexity measure is acceptable, only when its usefulness has been proven by a validation process. We evaluated and validated the *C(XSD)* metric both theoretically and empirically in section 4.2.4.1 and 4.2.4.2 respectively.

4.2.4.1 Theoretical Validation

The necessity for practical evaluation of any newly proposed metric is clear. For practical evaluation, in section 4.2.4.1.1, we examined our metric against a framework developed by Kaner [69] and in section 4.2.4.1.2 we referred Weyuker[73] properties (see section 3.2) for validation process.

4.2.4.1.1 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 *C(XSD)* metric, we followed the guidelines given by Kaner [69]. This approach for metric validation is more practical than the formal approach.

When we look at *C(XSD)* from the perspective given in [69], 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. Consequently, its scope of use is the software development group working specially for XML Web services [2, 3, 60, and 61].

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 4.2.2.

Measuring instrument to perform the measurement: It uses the instrument of *counting* by either human or by machine. For automated counting purpose, one can develop a program/tool for calculating the *C (XSD).*

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 [41], we find that, it is on the ratio scale.

Relationship between the attribute to the metric value: There is an inverse 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.

The Natural variability of readings from the instrument: Since the reading from our counting instrument is not subjective and does not require any interpretation, we can say that no variability (i.e. measurement error) on readings from the instrument can be expected. Note that, in case of automated counting, we assume that there is no bug in the devised algorithm.

Natural and foreseeable side effects of using the instrument: Once we automate the complexity calculation, it will not require considerable additional workload of manpower of the company. The only cost will be due to automation.

4.2.4.1.2 Evaluation of *C(XSD)* Metric by Weyuker's Properties

In the following paragraphs our metric *C(XSD)* has been evaluated against the Weyuker properties(see section 3.2) for establishing itself as a good and comprehensive measure and also to determine its strength and weaknesses.

Property 1: $(\exists P)(\exists Q)(|P| \neq |Q|)$. Where P and Q are program body. This property states that a measure should not rank all programs as equally complex. From Table 4.2 it can be easily observed that not all Schema files have the equal *C(XSD)* values i.e. there exist Schema files that have different complexity values measured by *C(XSD)* metric. Hence, this property is satisfied by *C(XSD)* metric.

Property 2: Let c be a non-negative number then there are only finitely many programs of complexity c. Taken c as a non-negative number. All Schema documents consist of only finite number of components (i.e. elements, attributes, complex types, simple types, and attribute and element groups). Now, as the *C(XSD)* measures the sum of the complexity weights of the components in a Schema document, some largest possible number can be assumed without harm that can be taken as an upper bound. Therefore, for a given number (complexity value), there are only finitely many Schema documents. Hence, *C(XSD)* holds this property.

Property 3: There are distinct programs P and Q such that $|P| = |Q|$. From Table 4.2 one can easily capture that there exist different Schema documents which have equal complexity values measured by *C(XSD)*, hence this property is satisfied.

Property 4: $(\exists P)(\exists Q)(P \equiv Q \ \& \ |P| \neq |Q|)$ .Consider the Schema file, shown in listing 4.4 [64] and its modified version shown in listing 4.5.

```
<?xml version="1.0" encoding="UTF-8" ?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/1998/Math/MathML"
 targetNamespace="http://www.w3.org/1998/Math/MathML"
 elementFormDefault="qualified">
 <xs:annotation>
  <xs:documentation>This is the XML schema module
  defining common attributes for the content part of MathML.
  Authors: Stéphane Dalmas, INRIA.
</xs:documentation>
  </xs:annotation>
 <xs:attributeGroup name="Definition.attrib">
  <xs:attribute name="encoding" type="xs:string" />
  <xs:attribute name="definitionURL" type="xs:anyURI" />
  </xs:attributeGroup>
</xs:schema>
```

Listing 4.4 The Schema document common-attrib.xsd.

Both Schema documents generate the same XML document, i.e. both Schema documents have the same functionality. *C(XSD)* value for the original Schema file shown in listing 4.4 is calculated by as:

$$C(XSD) = C(V_g) + C(G_g) + C(T_g)$$

$$= [C(E_g) + C(A_g)] + [C(EG_g) + C(AG_g)] + [C(cT_g) + C(sT_g)]$$

$$= \left[\sum_{i=1}^{N} we_i \, E_{g_i} + \sum_{j=1}^{M} wa_j \, A_{g_j}\right] + \left[\sum_{t=1}^{K} weg_t EG_{g_t} + \sum_{s=1}^{P} wag_s AG_{g_s}\right]$$

$$+ \left[\sum_{r=1}^{R} wc_r cT_{g_r} + \sum_{q=1}^{Q} ws_q sT_{g_q}\right]$$

$$= [0 + 0] + [0 + \sum_{s=1}^{1} wag_s AG_{g_s}] + [0 + 0]$$

$$= wag_{baseGroup} \pm \sum_{i=1}^{2} wa_i \, A_i$$

$$= 0 + wa_{encoding} + wa_{definitionURL}$$

$$= 1 + 1$$

$$= 2$$

The *C(XSD)* value for the modified Schema document shown in listing 4.5 can be computed by using the same formula and evaluated as 4. In the modified version of Schema document we only modified the attribute `encoding`. As explained in section 4.2.2 the weight value for an attribute or for an element is assigned based on the weight value of its type. In the modified version of Schema document the attribute `encoding` has user defined simple type `myString` and its weight value is evaluated as 3 according to the equation (13.1) (The weight value for `myString` is 3 according to the equation (17.2) since it is derived from built-in simple type and has two restrictions). Although we modified the attribute `encoding` its value space remains unchanged since we did not apply any restricting facet [31, 37, and 39] on its base type `Dstring` which is also derived from W3C XML Schema built-in simple type *string* by restriction without applying any facet. Hence, attribute `encoding` can take the same string value when XML document is generated by the two Schema files. This satisfies property 4.

```
<?xml version="1.0" encoding="UTF-8" ?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/1998/Math/MathML"
 targetNamespace="http://www.w3.org/1998/Math/MathML"
 elementFormDefault="qualified">
 <xs:annotation>
  <xs:documentation>This is the XML schema module
  defining common attributes for the content part of MathML.
  Authors: Stéphane Dalmas, INRIA.
</xs:documentation>
  </xs:annotation>
 <xs:attributeGroup name="Definition.attrib">
 <xs:attribute ref="encoding"  />
 <xs:attribute name="definitionURL" type="xs:anyURI" />
 </xs:attributeGroup>
 <xs:attribute name="encoding" type="myString"/>
 <xs:simpleType name="myString">
    <xs:restriction base="Dstring"/>
 </xs:simpleType>
 <xs:simpleType name="Dstring">
    <xs:restriction base="xs:string"/>
 </xs:simpleType>
</xs:schema>
```

Listing 4.5 The modified version of Schema document common-attrib.xsd.

Property 5: $(\forall P)(\forall Q)(|P| \leq |P;Q|$ *and* $|Q| \leq |P;Q|)$. This property states that the complexity of a program segments P and Q should each be less than or equal to the complexity of the composition of the two program segments; P;Q. Now, for example, consider the Schema documents with id 6 and 8 shown in Table 4.2. Each Schema file has different *C(XSD)* value. If we concatenate these two Schemas into one Schema file it is clear the complexity values for each of the two Schema file will be less than the complexity value of the resulting Schema complexity. This result shows that *C(XSD)* satisfies property 5.

Property 6: $(\exists P)(\exists Q)(\exists R)(|P| = |Q|)$ & $(|P;R|) \neq (|Q;R|)$. This property states that the resulting complexity of the composition of two program segments P:R is not necessarily the same as the complexity that results from the combination of two program segments Q;R, even though the complexity of P is equal to that of Q. By considering again Table 4.2 we can choose any two Schema files having equal complexity and one another Schema file that has different complexity from the two. Let take the Schema files with id numbers 4, 5, and 6 that represents P, Q, and R. The *C(XSD)* values of the two Schema files with id numbers 4 and 5 are equal which is 12 and that of the Schema with id 6 is 32. If we combine the Schema files with id numbers *4* and *6* into a new Schema the complexity value of the resulting Schema file is equal to the complexity value of the Schema file that is the combination of the Schema documents having id numbers 5 and 6. Therefore this property is not satisfied by *C(XSD)* metric.

Property 7: There are program bodies P and Q such that Q is formed by permuting the order of the statements of P, and $(|P| \neq |Q|)$. The place of the declaration/definition of the components in the Schema cannot change the *C(XSD)* value since we consider internal architectures of each component while we are calculating *C(XSD)* for a given Schema document. Therefore, this property is satisfied by the *C(XSD)* metric.

Property 8: If P is renaming of Q, then $|P| = |Q|$. The value of *C(XSD)* is an integer so renaming of a program cannot change the value of *C(XSD)* and as such this property is clearly satisfied by the given complexity measure.

Property 9: $(\exists P)(\exists Q)(|P| + |Q|)<(|P;Q|)$. For a given two Schema files, P and Q, the only way for the two Schema files to be combined is to concatenate them into one

Schema document. It is clear that the resulting Schema document's complexity value, *C(XSD)*, is equal to the summation of each of the Schema documents' complexity values. Therefore, this property is not hold by *C(XSD)*. However, the usefulness of this property is the topic of research. Misra [89] has modified it and suggest that if $(\exists P)(\exists Q)(|P| + |Q|) \leq (|P;Q|)$, it will be more valuable in evaluating the complexity metric.

Our measure satisfied the modified Weyuker's 9th property [89]. It also proves the additivity nature of the measure. In other words we can say that our *C(XSD)* metric is on ratio scale.

4.2.4.2 Empirical Validation of *C(XSD)*

Empirical validation proves the practical utility of a new metric. For the empirical validation of the proposed metric we analyzed 70 real example Schema documents from the web. Note that most of these analyzed Schemas are extracted from WSDL [42] documents (the Schema definition is given under <types> element of WSDL document). We applied the *C(XSD)* and *#CT* metric to these analyzed Schemas and the corresponding values are given in Table 4.2. We have assigned each Schema document with id numbers for the sake of clarity and the references to web links for these files are shown in Table 2 in Appendix C with the same id numbers. In Table 4.2 *#CT* is the total number of global and local complex types; *#RE* is the number of global recursive elements; *C(XSD)* value for Recursive Schema documents is given in the last column where R=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. The value for the *#CT* [16, 17 and 38] metric is calculated by summing up all global and local complex type definitions of related XSDs. We have demonstrated in section 4.2.3, how to calculate the complexity of each components of the example Schema file shown in listing 4.3 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.

From Table 4.2 it can be observed that even some of Schema files have equal *#CT* values they do not have equal *C(XSD)* values. The *#CT* metric cannot reflect the actual complexity of these Schemas since it neglects the recursive structures of XSD's components. Consider for example the #CT value for the Schema document with id 70 which is 69. Based on this value Schema document with id 70 should be the most complex one among the other XSDs. However when the complexities of these XSDs are measured by *C(XSD)* metric it is found that the most complex Schema is the Schema with id 59 due to its recursive components among the other XSDs. As we suggested earlier having greater number of recursive elements does not always result in for a given Schema to have higher complexity.

Table 4.2. The analyzed Schema files for the empirical validation of *C(XSD)* metric.

#ID	*#CT*	*C(XSD)*	*#RE*	*R=2*
1	*3*	*8*		
2	*4*	*7*		
3	*4*	*11*		
4	*4*	*12*		
5	*5*	*12*		
6	*5*	*32*		
7	*6*	*36*		
8	*8*	*28*		
9	*8*	*22*		
10	*8*	*50*		
11	*8*	*20*		
12	*8*	*89*		
13	*8*	*18*		
14	*9*	*19*		
15	*10*	*48*		
16	*10*	*37*		
17	*11*	*18*		
18	*11*	*70*		
19	*11*	*18*		
20	*12*	*29*		
21	*12*	*49*		
22	*12*	*63*		
23	*13*	*163*	*4*	*147+8*R*
24	*14*	*131*	*7*	*103+14*R*
25	*14*	*30*		
26	*14*	*30*		

Table 4.2 (Cont.)

27	*14*	*29*		
28	*14*	*23*		
29	*14*	*29*		
30	*14*	*73*		
31	*15*	*61*		
32	*15*	*164*		
33	*16*	*54*		
34	*17*	*47*		
35	*17*	*44*		
36	*17*	*118*		
37	*18*	*52*		
38	*18*	*88*		
39	*18*	*70*		
40	*18*	*94*		
41	*18*	*179*		
42	*18*	*582*	*16*	*518+32*R*
43	*19*	*116*		
44	*20*	*93*		
45	*20*	*94*		
46	*21*	*55*		
47	*21*	*64*		
48	*21*	*50*		
49	*22*	*35*		
50	*22*	*60*		
51	*22*	*32*		
52	*23*	*54*		
53	*24*	*113*		
54	*24*	*47*		
55	*24*	*334*		
56	*25*	*532*		
57	*26*	*144*		
58	*27*	*275*	*2*	*271+2*R*
59	*28*	*867*	*18*	*741+63*R*
60	*30*	*108*		
61	*30*	*112*		
62	*30*	*162*		
63	*33*	*221*		
64	*34*	*83*		
65	*38*	*202*		
66	*39*	*405*		
67	*41*	*493*	*31*	*259+117*R*
68	*43*	*840*	*66*	*166+337*R*
69	*54*	*134*		
70	*69*	*349*		

This suggestion can be supported by comparing the *C(XSD)* values of the Schema documents with id 59, 67 and 68. The number of recursive elements is 18, 31 and 66 respectively and their *C(XSD)* values are 867, 493, 840. As can be seen that even though the Schema with id 68 has greatest number of recursive elements its C(XSD) value is less than that of the Schema with id 59. Another suggestion stated earlier was that a given Schema that contains recursive elements may not always has higher complexity than that of a Schema document that has no recursive elements. From Table 4.2 it can be observed that this suggestion is supported by making comparison between the values of *C(XSD)* for the Schema documents that contains no recursion and that of Schemas containing recursive elements.

The graph depicted in Figure 4.1 is drawn based on the data given in Table 4.2. Note that the R value is assigned to 2 for calculating *C(XSD)* values of recursive Schema documents and the data is ordered by *#CT* values. As can be seen from Table 4.2 and Figure 4.1 *C(XSD)* metric evaluates different values for Schemas having equal *#CT* values. Consider the complexity values of XSDs with id numbers 37, 38, 39, 40, 41, 42 in Table 4.2. Although these XSDs have the same #CT values which is 18 their *C(XSD)* values are not the same.

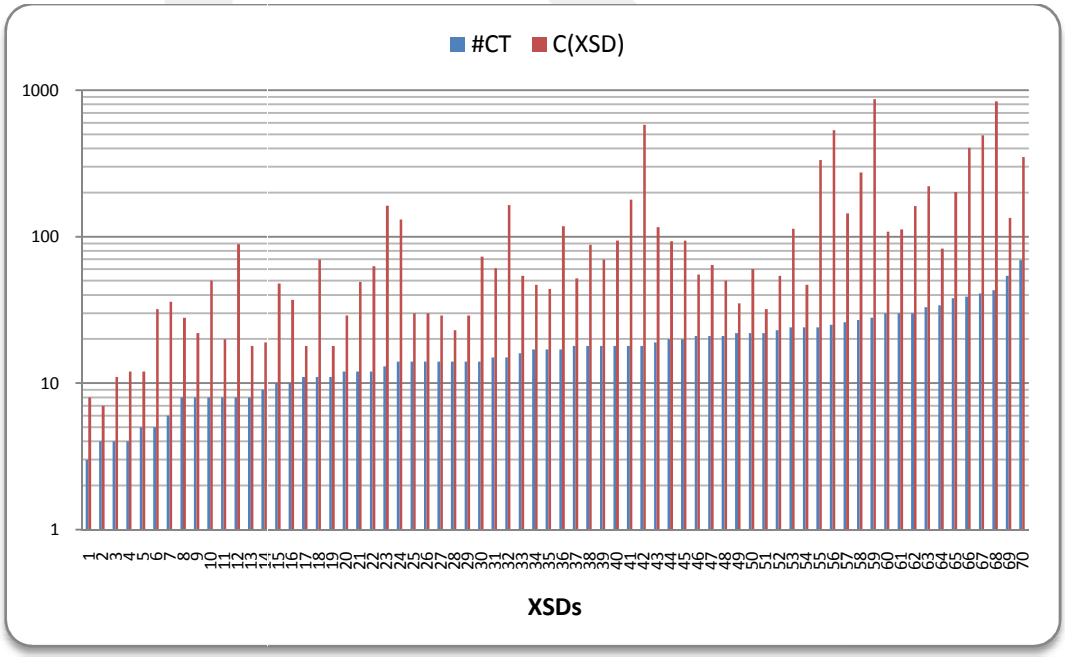

Figure 4.1 *#CT* metric vs. *C(XSD)* metric.

Further, these *C(XSD)* values increase very dramatically as can also be observed from Figure 4.1. The reason behind this increase is that *#CT* metric does not consider

internal architectures of complex types which may include recursion and the resulting complexity. On the other hand *C(XSD)* metric is able to capture this complexity very apparently. From Figure 4.1 one can easily see that as *#CT* metric values increase smoothly *C(XSD)* values do not increase accordingly.

In order to show our C(XSD) metric is more distnictive in measuring complexity of XSD than #CT metric we compared #CT values and C(XSD) values for analyzed XSDs.

Table 4.2.1 Count of #CT metric values for analyzed Schema documents

Count of #CT Values	
#CT Value	Count of XSD
14	7
8	6
18	6
4	3
11	3
12	3
17	3
21	3
22	3
24	3
30	3
5	2
10	2
15	2
20	2
3	1
6	1
9	1
13	1
16	1
19	1
23	1
25	1
26	1
27	1
28	1
33	1
34	1
38	1
39	1
41	1
43	1
54	1
69	1
Grand Total	70

For comparison we counted number of analyzed XSDs for which #CT evaluates the same value, (see table 4.2.1) . Similarly, we also counted number of analyzed XSDs for which C(XSD) evaluates the same value, (see table 4.2.2).It can be seen in the table 4.2.1 and table 4.2.2 the number of the XSDs for which #CT evaluates the same value is greater than that of C(XSD). For example, in table 4.2.1, #CT the value 14 is calculated for 7 of the 70 different XSDs and the values 8, 18 (both) are calculated for 6 of the total XSDs.

Table 4.2.2 Count of C(XSD) metric values for analyzed Schema documents

Count of C(XSD) Values	
C(XSD) Value	Count of XSDs
18	3
29	3
12	2
30	2
32	2
47	2
50	2
54	2
70	2
94	2
7	1
8	1
11	1
19	1
20	1
22	1
23	1
28	1
35	1
36	1
37	1
44	1
48	1
49	1
52	1
55	1
60	1
61	1
63	1
64	1
73	1
83	1
88	1
89	1
93	1

Table 4.2.2 (Cont.)	
108	1
112	1
113	1
116	1
118	1
131	1
134	1
144	1
162	1
163	1
164	1
179	1
202	1
221	1
275	1
334	1
349	1
405	1
493	1
532	1
582	1
840	1
867	1
Grand Total	70

On the other hand, in table 4.2.2 C(XSD) evaluation only 3 XSDs out of 70 are calculated as 18 which has the maximum repetition count in C(XSD) evaluation. In more detail, #CT distributed the 70 Schemas on the scale of 0-69 evaluation values in which only 19 of them judged unique evaluation value, while C(XSD) distribute the same Schemas on the scale of 0-867 evaluation values in which 48 of them each evaluated to different value. With this figures for the 70 analyzed Schema examples, it can be interpreted that C(XSD) metric evaluation is more distinctive than #CT metric and eventually, it is more sensible.

4.2.5 Concluding Remark on *C(XSD)* Metric

The *C(XSD)* metric is evaluated on the basis of the internal complexities of major building components of XSDs and computed by using the provided formulas. In section 4.2.3 we have demonstrated that the internal architecture of XSDs' building components affect the overall complexity of XSD. In order to check the reliability of

the *C(XSD)* metric, it is theoretically evaluated and empirically validated in sections 4.2.4.1 and 4.2.4.2 respectively. The *C(XSD)* 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, and recursive components, an important issues, which have not been considered yet by other researchers. Empirical validation of *C(XSD)* metric verifies that *C(XSD)* metric can be a good indicator for measuring complexities of XSDs. Hence, in terms of their complexity values, XSDs having equal *#CT* can easily be differentiated by using *C(XSD)* metric.

As a consequence we can conclude that *C(XSD)* metric can provide useful feedback for the assessment and improvement of the quality of XSDs in terms of maintainability.

4.3 Schema Entropy Metric (*SE*)

Our second metric for measuring physiological complexities of Schema documents is based on entropy concept and exploit the directed graph representation of XSDs.

The entropy adapted from communication information theory of Shannon [52] is defined as a measure of uncertainty or variety. According to Shannon's definition if M is the message set (M_1, M_2, ...,M_n) from which the message consist of, the self-information of the i^{th} message M_i with probability of occurrence $P(M_i)$ is:

$$I(M_i) = \log_2 P(M_i)$$

The probability $P(M_i)$ is of i^{th} most frequently occurring message in a conveyed message set M consisting of N number of messages and can be defined as :

$$P(M_i) = f_i/N$$

where f_i is the number of occurrences of i^{th} most frequently occurring message.

Then, the value of entropy can be calculated by averaging the self information over all messages in M message set. Thus, entropy is given by:

$$H=-\sum_{i=1}^{n} P(M_i)\log_2 P(M_i)$$

The entropy concept has been applied by many researchers [44-51, 53-58] for the assessment of the complexity of software products that are developed by using procedural or OO programming technologies. These measures were developed by

adapting Shannon's entropy theory [52] as a measure of uncertainty or variety. Davis and LeBlanc [44] used entropy that was adopted from Shonnon's notion of entropy to assess syntactic complexity of FORTRAN and COBOL code. In their work the software code was partitioned to "*chunks*" that was defined as a group of related items which can be formed into single mental concepts. The entropy was used to measure three aspects of chunks represented by a graph in a program: the structure of chunk connections, chunk content and chunk size. In order to use entropy metric as a structure measurement LeBlanc *et al*. constructed a graph that exhibits the data dependency between these chunks. The chunks were distinguished by the number of their incoming and outgoing edges in the graph representation of a program. From this graph it was observed that a graph consisting of a number of similar substructures tend to have lower entropy than the other graph not having such regularity although these two graphs have the same number of nodes. The result drawn from this observation is that entropy is potentially more useful measure than McCabe's V(G) [71] since entropy takes into account for greater variety in structure brought by nesting, though this variety is ignored by V(G).

The use of entropy as a measure of information content was also introduced by Harrison [45]. Harrison's software complexity metric is based on empirical program entropy that allows the measurement of the information content of procedural programs. Torress and Samadzadeh [46] showed that software reusability has an inverse relationship with entropy. In all of these works the usage of entropy to evaluate code complexity was based on Zweben and Halstead [58] work which shows that "*operators*" are distributed with a natural probability, hence, could be used to measure a product's information content. Bansiya *et al*. [57] used "*name strings*" to measure *Class Definition Entropy* (*CDE*) assuming that all "*name strings*" represent approximately equal information are related to the possible error insertion by misusing the string.

However, the usage of entropy as a complexity metric has not yet been extended and validated for the assessment of XML Schema documents. The *SE* metric is intended to measure the complexity of XSD document due to diversity in the structures of its elements and is established by following similar approach that was taken by Davis and LeBlanc [*44*].

4.3.1 Motivation

In section 4.2 we presented a complexity metric *C(XSD)*[67] for the assessment of the quality of XML Schema and complexity value for a given XSD is evaluated by considering all of its components' complexities. In this evaluation each component's complexity degree is reflected by a weight value which is assigned based on component's internal architecture. While assigning a weight value for each component of Schema document we observed that calculation of weight values is easier when most Schema components having similar structures appear more frequently and is harder when Schema has mostly diverse structured elements. In other words, in the Schema document the occurrence of similar structured components with high frequency made the calculation easier due to gained familiarity. In addition, it was also easier to understand and remember the structures of the Schema components when most of Schema elements have the same structure. Although numbers of measures have been developed (see section 1.2) for the assessment of the quality of Schema documents none of them has not considered yet measuring the variety in Schema's elements structure and resulting complexity. These observations lead us to make a suggestion that the more repetitions of the similar structured elements makes XSD easier to understand. In order to support our suggestion we developed Schema Entropy *(SE)* metric which is based on the entropy concept from information theory [52]. We believe that the *SE* metric can differentiate XSDs in terms of their physiological complexities due to the diversity and repetitions in their elements' structures. Since it takes into account the repetition of similar structured elements, *SE* metric can capture decreasing physiological complexity of XSD due to familiarity gained by navigating the similar structures many times. It is our opinion that the *SE* metric can be useful to evaluate and maintain the quality of XML Schema document in terms of its maintainability.

The *SE* metric exploits a directed graph representation of a Schema document, *G(XSD)*. Therefore before giving the definition of *SE* metric we first explain how to represent a given Schema document by a directed graph and classification of its elements according to their structures in section 4.3.2. The definition and the approach followed for the establishment of *SE* metric as a complexity measure for Schema document and, its demonstration is given in section 4.3.3 and 4.3.4

respectively. The practical applicability of *SE* is checked through empirical and theoretical validations which are explained in section 4.3.5.

4.3.2 Directed Graph Representation of XSD and Equivalence Classes

The directed graph representation of Schema document, G(XSD) can be defined as $G(XSD) = (N, E)$, where N is a set of nodes representing the elements of XSD and can be defined as $N = <N_1, N_2,..., N_m>, m = 1,2...n$, *n* is the total number of element definitions in XSD; E is a set of edges that represent parent-child relationships between elements of XSD. The elements of XSD that have no child elements are represented by leaf nodes and the attributes that an element of XSD has are listed in square shapes and are connected to their associated element nodes having circle shape by straight lines in *G(XSD)*. Any particular node, N_i, in *G(XSD)* representing i^{th} element of XSD, can be identified according to the number of incoming edges from other nodes to the N_i; the number of outgoing edges originated from N_i to other nodes and the number of attributes that an element i has.

The counts of incoming and outgoing edges of an element node can be measured by *fan-in* and *fan-out* metrics [16, 18, 19] respectively. Note that *fan-in* and *fan-out* measures were originally proposed by Kafura [72] to measure total level of information flow between individual modules and the rest of the system.Identification of node N_i in G(XSD) can be provided by the triple: $N_i = <fin_i, fout_i, s_i>$ where fin_i and $fout_i$ are the counts of its incoming and outgoing edges, and s_i is the number of attributes of element of XSD represented by N_i. Note that the total *fan-in* value is always equal to total *fan-out* value of G(XSD). Any two elements *k* and *m* of XSD represented by the nodes N_k and N_m in G(XSD) are equal in structure and have the same complexity only if $fin_k = fin_m$, $fout_k = fout_m$ and $s_k = s_m$.

As an example, consider the Schema document books.xsd shown in listing 4.6 and its directed graph depicted in Figure 4.2. From Figure 4.2 it can be observed that not all the nodes have the same structures i.e. the number of their incoming and outgoing edges and, the number of attributes is not equal. For instance, the node representing the author element is identified by <1, 2, 0> and the node representing the publisher element is identified by <1, 2, 1> and, although both elements have equal *fan-in* and

fan-out values they do not have same number of attribute. Hence these two nodes i.e. Schema element definitions can be distinguished by the difference in their structures.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="books">
  <xsd:complexType>
   <xsd:sequence maxOccurs="unbounded">
    <xsd:element name="book">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author">
         <xsd:complexType>
          <xsd:sequence>
           <xsd:element name="firstname" type="xsd:string"/>
           <xsd:element name="lastname" type="xsd:string"/>
          </xsd:sequence>
         </xsd:complexType>
       </xsd:element>
       <xsd:element name="publisher">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:element name="firstname" type="xsd:string"/>
          <xsd:element name="lastname" type="xsd:string"/>
         </xsd:sequence>
         <xsd:attribute name="email" type="xsd:string"/>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="ISBN" type="xsd:positiveInteger"/>
      <xsd:attribute name="date" type="xsd:date"/>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Listing 4.6 The schema documents books.xsd.

Distinguished the nodes in *G(XSD)* and elements accordingly in terms of the difference in their structures, the nodes having similar structures are grouped into the same equivalence classes. More formally, an equivalence class C having m number

of elements can be defined as a set of similar structured element nodes in *G(XSD),* that is,

$C = <N_0, N_1,...N_m>$,where the element nodes $N_0 = N_1 = N_2...N_{m-1} = N_m$.

The number of equivalence classes reflects number of unique element structures in XSD and the member counts of each class reflect the number of occurrences of each class member. Each element that belongs to the same class has the same structure that is, their *fan-in*, *fan-out* and number of attributes are equal, but the elements belonging to different classes have distinct structure. For instance, the equivalence classes of books.xsd given in listing 4.6 and projects.xsd given in listing 4.7 are exploited from their corresponding graphs (see Figure 4.2 and Figure 4.3) and are listed in listing 4.8 and listing 4.9 respectively. Note that both Schemas have equal number of elements which is nine and while books.xsd has five distinct structured elements reflected by the number of its equivalence classes, it is six for projects.xsd.

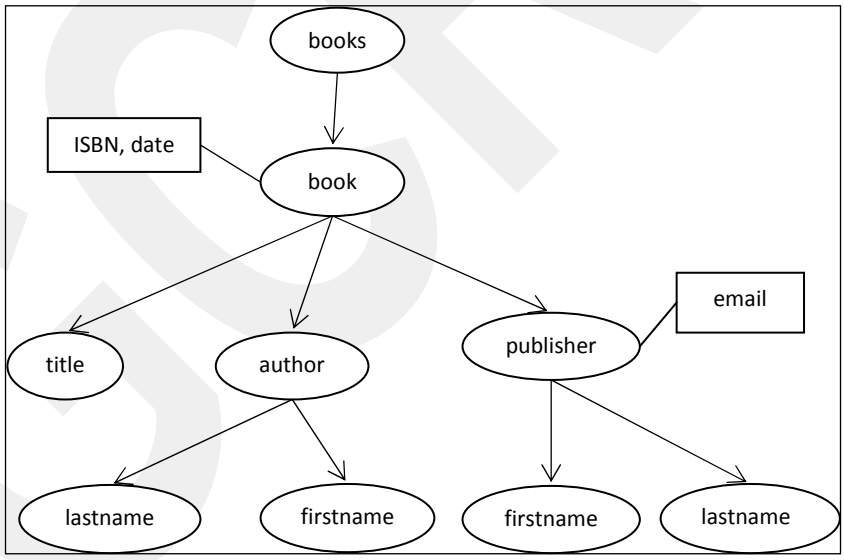

Figure 4.2 The directed graph representation of schema books.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="projects">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="project">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="scripts">
        <xs:complexType>
         <xs:sequence>
          <xs:element maxOccurs="unbounded" name="script">
           <xs:complexType>
            <xs:sequence>
              <xs:element name=”language” type="xs:string" />
            </xs:sequence>
            <xs:attribute name="scriptname" type="xs:string" />
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="namespaces">
        <xs:complexType>
         <xs:sequence>
          <xs:element maxOccurs="unbounded" name="namespace" type="xs:string" />
        </xs:sequence>
        <xs:attribute name="url" type="xs:string" />
       </xs:complexType>
      </xs:element>
      <xs:element name="singletons">
       <xs:complexType>
        <xs:sequence>
         <xs:element maxOccurs="unbounded" name="singleton">
          <xs:complexType mixed="true">
           <xs:attribute name="createdBy type="xs:string" />
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
    </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Listing 4.7 The Schema document projects.xsd.

When the elements of books.xsd are classified according to their *fan-in, fan-out* and number of attributes we evaluate 5 distinct structured elements i.e. equivalence classes given in listing 4.8 and the frequencies of each distinct structured element of books.xsd are 1, 1, 5, 1 and 1. In listing 4.8 the class C_3 has five elements that are equal in structure and the remaining classes have only one distinct structured element. This means that one element structure appears five times in books.xsd and has higher frequency among the other four distinct element structures. By using the same classification method we can find that the Schema projects.xsd has 6 distinct structured elements and its equivalence classes are shown in listing 4.9. In listing 4.9 the member counts of class C_3 is three and that of class C_4 is two and, all the remaining classes has only one distinct structured element. So, the frequencies of each distinct structure of projects.xsd are 1, 1, 3, 2, 1, and 1.

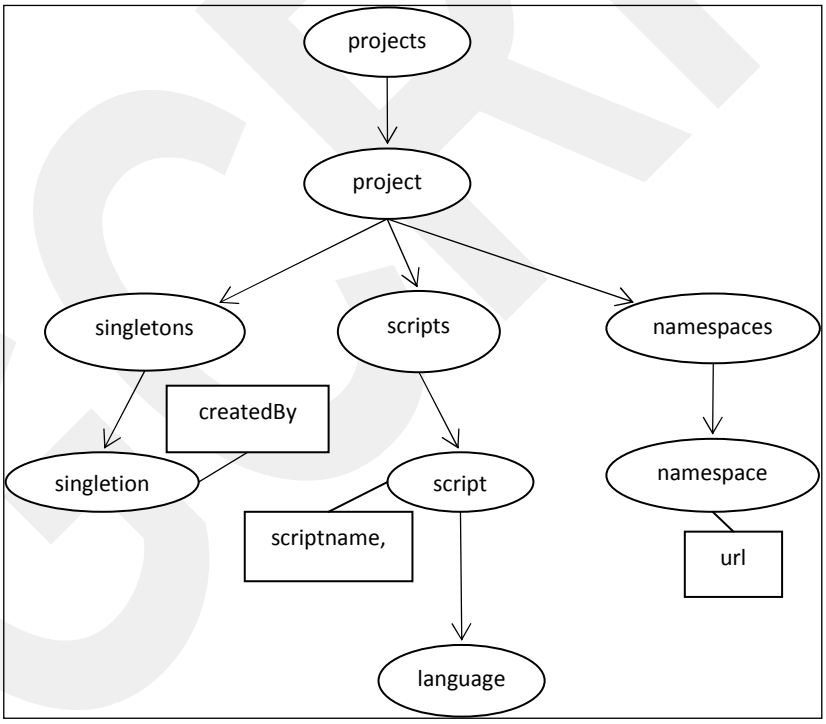

Figure 4.3 The directed graph representation of the Schema projects.xsd.

C_1= {books},
C_2= {book},
C_3= {title, firstname, lastname, firstname, lastname},
C_4 = {author},
C_5 = {publisher}

Listing 4.8 Equivalence classes of the Schema document books.xsd.

C_1= {projects},
C_2= {project},
C_3= {singletions, namespaces, scripts},
C_4 = {singletion, namespace},
C_5 = {script},
C_6 = {language}

Listing 4.9 Equivalence classes of the Schema document projects.xsd.

4.3.3 Definition of Schema Entropy Metric (*SE*)

By following similar approach taken by Davis and LeBlanc [44] and based on their definitions we applied entropy metric to measure complexity of XSD files. In our case the declarations of the components of XSD can be seen as "chunks" since these components can be related with the other elements in the form of parent-child relationship that can be shown in a directed graph representation of Schema document *G(XSD)*. In *G(XSD)* while *fan-in* metric's value gives information about how many times an element node is referenced by the other element nodes, i.e. the number of parent nodes of a child node, the *fan-out* metric gives the number of element nodes that a particular element node depends on, i.e. number of child nodes. While low *fan-out* for any parent node can be interpreted as low dependency on a few child nodes, high *fan-in* for any child node can be interpreted that many parent nodes are dependent on that child node. It is clear that having high *fan-in* for a particular child node implies that that child node has impact on its parent nodes since these parent nodes will be affected by any modification made in the child node. On the other hand, for any particular node, zero *fan-in* and *fan-out* means that neither it has effect on any other nodes and nor is affected by the other nodes. Hence, both *fan-in* and *fan-out* metrics can measure the dependency between nodes and the nodes that have the equal *fan-in*, *fan-out* values and equal number of attributes can be

treated as equal in terms of their structural complexities. As stated in section 4.3.2, any two elements k and m of XSD represented by the nodes N_k and N_m in *G(XSD)* are equal in structure and have the same complexity only if $fin_k = fin_m$, $fout_k = fout_m$ and $s_k = s_m$.

Based on the entropy definition stated in section 4.3 the entropy of a given Schema document having *n* distinct class, (C_n), of elements can be calculated using relative frequencies as unbiased estimates of their probabilities $P(C_i)$, $i=1,2,\ldots,n$.

The relative frequency of occurrence of the equivalence classes of the Schema document is the number of elements inside the equivalence class divided by the total number of elements in the Schema document. As an example, for projects.xsd the relative frequency of occurrence of its equivalence class C_3= {singletions, namespaces, scripts} is $P(C_3) = 3/9$.

When all elements of XSD are placed in the same equivalence class the minimum entropy value is evaluated. Since there is only one class, i.e. n=1, and all elements are grouped in this class the relative frequency of occurrence of that class, $P(C_1)=1$, and *SE* value is:

$$\begin{aligned} SE &= -\sum_{i=1}^{1} P(C_i)\log_2 P(C_i) \\ &= P(C_1)\log_2 P(C_1) \\ &= 0 \end{aligned}$$

On the other hand the possible maximum entropy occurs when each element of XSD has different *fan-in, fan-out* values and number of attributes i.e. each has different structure. The *SE* value of that Schema, in this case is:

$$\begin{aligned} SE &= -\sum_{i=1}^{n} P(C_i)\log_2 P(C_i) \\ &= -\sum_{i=1}^{n} (1/n)\ \log_2 (1/n) \\ &= \log_2 n \end{aligned}$$

where n is the number of equivalence classes.

4.3.4 Illustration of *SE* Metric

In order to demonstrate *SE* metric we used two example Schema documents books.xsd and projects.xsd that are given in section 4.3.2 and shown in listing 4.6 and listing 4.7 respectively. The diversity in their elements' structures is measured via our metrics that consider the equivalence classes of these XSDs (see listing 4.8

and listing 4.9) and the number of occurrences of each different structured element i.e. unique structured element in each class.

Example 4.3: The entropy value for books.xsd , *SE*, is :

$$SE_{books.xsd} = -\sum_{i=1}^{5} P(C_i)\log_2 P(C_i)$$
$$= -[(1/9)*\log_2(1/9)+(1/9)*\log_2(1/9)+(5/9)*\log_2(5/9) + (1/9)*\log_2(1/9) + (1/9)*\log_2(1/9)]$$
$$= 1.87996$$

The *SE* value for projects.xsd is:

$$SE_{projects.xsd} = -\sum_{i=1}^{6} P(C_i)\log_2 P(C_i)$$
$$= -[(1/9)*\log_2(1/9) + (1/9)*\log_2(1/9) + (3/9)*\log_2(3/9) + (2/9)*\log_2(2/9) + (1/9)*\log_2(1/9) + (1/9)*\log_2(1/9)]$$
$$= 2.41938$$

As can be seen from the example 4.3 although both example XSDs have the same number of element definitions they do not have equal *SE* values. This is due to the fact; the Schema documents with lower *SE* tend to be dominated by fewer distinct structured elements that have higher frequency of occurrences. In books.xsd the number of equivalence classes reflecting number of unique structures is five and, in projects.xsd it is six. These numbers imply that compared with books.xsd the Schema projects.xsd has more diversity in its elements' structures. Further, by looking the class C_3 of books.xsd it can be observed that five elements has equal in structure, that is the same structure appears five times. On the other hand, in projects.xsd maximum repetition of similar structured elements is three which is the member counts of class C_3. It will be more obvious that books.xsd has less diversity when we calculate the ratio of the number of distinct i.e. unique structured elements to total elements of XSD; we find it as 5/9 for books.xsd and 6/9 for projects.xsd. From these ratios we can guess which XSD will have lower *SE*. However, the this ratio values fail in differentiating any two XSDs in terms of their complexities when both XSDs have equal number of classes, i.e. unique structured elements and have same number of elements since frequencies of each distinct structured element are ignored. As larger XSDs in terms of elements' count generally have more elements, *SE* measure may tend to be lower. However, this reflects the notion that larger XSDs

are able to contain more repetition, and so *SE* reflects this. It is likely that real world XSDs will not have *SE* measurements approaching minimum value which is 0 and maximum value $\log_2(n)$, where *n* is the total number of elements of XSD. As a consequence, *SE* metric can be useful in comparing XSDs having equal number of elements.

As stated earlier the structural varieties of elements declared in Schema has not been captured by any existing Schema measures. Therefore, usefulness of *SE* metric can be verified by comparing it with the other XSD complexity measures such as *element fanning* [16, 18, and 19], number of elements [17]. The *element fanning* for a given Schema is calculated as follows:

$$Fanning = e/n$$

where ***e*** is the number of edges i.e. total *fan-in* or *fan-out* and ***n*** is the number of nodes in the directed graph of the Schema. The higher *fanning* value for a Schema can be interpreted as elements are highly connected i.e. dependent to each other thus modification made in any individual element will update the other element to which that individual element is connected. Hence, higher value of *fanning* metric indicates higher complexity. The #E [17] metric is a kind of size metric which measures the total number of local and global element definitions/declarations in XSD.

From Table 4.3 it can be observed that the values of complexity measures, the *fanning*, *#E* and *C(XSD)* are consistent with each other. However, *SE* metric has different values for the two Schemas as a measure of their complexities. It is due to the fact; the *SE* metric considers the diversity in the structures of each element appearing in the graph representation of Schema document and their frequencies. The Schema documents that exhibit greater variety in structures of elements with less frequency of occurrences have the greater value of *SE* than the Schema documents that exhibit less variety in the structures of elements with high frequency of occurrences i.e. more repetitions of similar structured elements. Since the high frequency of similar structured elements makes the developer more familiar to the Schema structure overall understandability of Schema document becomes much easier. This was neglected by the other compared complexity metrics.

The *SE* metric is also useful for comparing XSDs that conforms the same resulting XML documents. Consider for example the Schema document books2.xsd (see listing 4.10) which is the modified version of books.xsd and, its graph representation is depicted in Figure 4.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="books">
  <xsd:complexType>
   <xsd:sequence maxOccurs="unbounded">
    <xsd:element name="book">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="title" type="xsd:string"/>
       <xsd:element name="author">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:group ref="names"/>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="publisher">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:group ref="names"/>
         </xsd:sequence>
         <xsd:attribute name="email" type="xsd:string"/>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="ISBN" type="xsd:positiveInteger"/>
      <xsd:attribute name="date" type="xsd:date"/>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
<xsd:group name="names">
         <xsd:sequence>
          <xsd:element name="firstname" type="xsd:string"/>
          <xsd:element name="lastname" type="xsd:string"/>
         </xsd:sequence>
</xsd:group>
</xsd:schema>
```

Listing 4.10 The schema document books2.xsd, a modified version of books.xsd.

In this modified version we defined a reusable global elements group names. By giving reference to this global elements group we defined the firstname and lastname elements' within author and publisher elements. The equivalence classes of modified version books2.xsd are given in listing 4.11.

C_1= {books},

C_2= {book},

C_3= {firstname, lastname},

C_4 = {author}

C_5 = {publisher}

C_6 = {title}

Listing 4.11 Equivalence classes of the Schema document books2.xsd.

And the *SE* value for books2.xsd is:

$$SE_{books2.xsd} = -\sum_{i=1}^{6} P(C_i) log_2 P(C_i)$$
$$= -[(1/7)*\log_2(1/7) + (1/7)*\log_2(1/7) + (2/7)*\log_2(2/7) + (1/7)*\log_2(1/7) + (1/7)*\log_2(1/7) + (1/7)*\log_2(1/7)]$$
$$= 2.80735$$

The values of complexity measures for this new design of Schema document books2.xsd are shown in the last line of Table 4.3. When the ratio of distinct structured elements to the total number of elements appearing in the graph representation of books2.xsd is calculated it is found as 6/7 which is greater than 5/9, the ratio calculated for books.xsd, thus it is expected that the complexity of books2.xsd will increase since the number of distinct structured elements increases, i.e. diversity in structures is higher. Note that the ratio does not consider the repetition of similar structured elements. As can be observed from this Table, by adding reusable component to the modified version of books.xsd its complexity value is increased since repetition of similar structured elements is decreased and variety in structures is increased. The side effect of increasing number of reuse of same components is that increasing number of affected components that use the same reusable component. That is, when we modify the definitions of firstname and lastname elements and make them global, both author and publisher elements of books2.xsd will be affected since the number of reuse for firstname and lastname will be two in this case. On the contrary, when firstname and lastname elements are declared locally inside author and publisher elements as declared in books.xsd, the number of their use is only one; hence any modification made on one of these pair

only affects one of the associated parents. As a consequence, we may suggest that increasing reusability in XSD components may result in increasing complexity due to increasing number of affected components. However, in order to verify this result it would be required to make much more research on the effects of reusable components on the complexity of Schema documents, which is one of our future work.

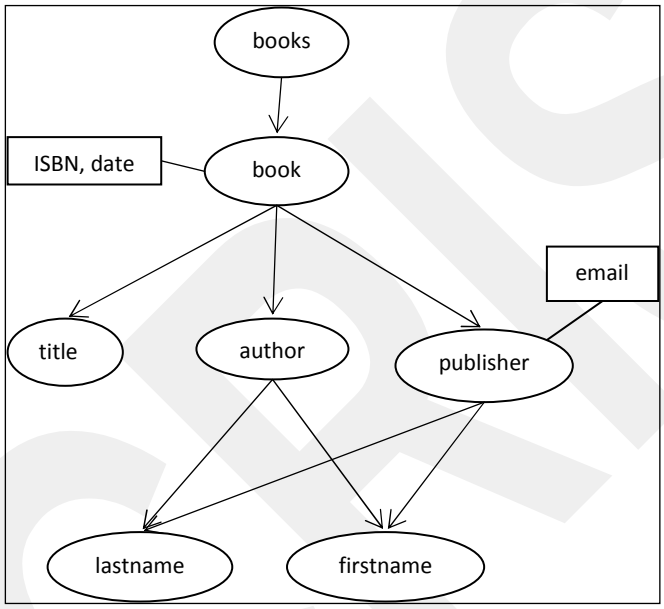

Figure 4.4 The directed graph representation of books2.xsd.

Table 4.3 The values of complexity measures developed for example XSDs.

Schema	*Listing*	*Figure No*	*Fanning*	*#E*	*C(XSD)*	*SE*
books.xsd	*4.6*	*4.2*	*8/9*	*9*	*12*	*1.87996*
projects.xsd	*4.7*	*4.3*	*8/9*	*9*	*12*	*2.41938*
books2.xsd	*4.10*	*4.4*	*8/7*	*7*	*12*	*2.80735*

4.3.5 Validation of *SE* Metric

The practical applicability of *SE* is checked through theoretical and empirical validations. As a part of theoretical validation process, in section 4.3.5.1 the *SE* metric is examined against the practical framework developed by Kaner[69] which we have used for checking our *C(XSD)* metric in section 4.2.4.1.1, and for checking *DSERS* metric which will be introduced in section 4.4. In section 4.3.5.2 we evaluated *SE* metric by using Weyuker's[73] principles in order to show the natural scale for it to be applicable. For empirical validation of *SE* metric we have analyzed 40 Schema documents. Note that most of the analyzed Schemas were extracted from WSDL (Web Service Description Language)[3, 42, 61] documents and these WSDL documents were downloaded from well known web sites such as *xmethods.com*, *webservicex.net* which provide link to Web services. The statistic we have collected after analyzing these Schemas is shown in Table 4.4. The empirical validation of *SE* metric is given in section 4.3.5.3.

4.3.5.1 Practical Evaluations of *SE* Metric

When we look *SE* from the perspective given in [69], *DSERS* 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 metrics 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 after coding. Consequently, its scope of use is the software development group working specially for XML Web services [2, 3, 60, and 61].

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 metrics have been defined formally in section 4.3.3.

Measuring instrument to perform the measurement: It uses the instrument of *counting* by either human or by machine. For automated counting purpose, one can develop a program to calculate the complexity value.

Natural scale for the metric: For the natural scale for our measure, we have to go through measurement theory [41]. When we analyze our measure according to Morasca [41], we find that, it is on the ratio scale.

Relationship between the attribute to the metric value: There is an inverse relation between the quality of the Schema and our metric *SE*. If the *SE* value decreases, it is clear that the Schema quality will increase. This is because lower *SE* can be interpreted that the developer makes less effort to understand and remember the structures of elements and dependency between them due to gained familiarity with elements having higher frequency of occurrences. Note that *SE* is not the unique indicators of Schema quality and the same argument is true for the relation between the *SE* value and the developer quality attribute.

The Natural variability of readings from the instrument: Since the reading from our counting instrument is not subjective and does not require any interpretation, we can say that no variability (i.e. measurement error) on readings from the instrument can be expected. Note that, in case of automated counting, we assume that there is no bug in the devised algorithm.

Natural and foreseeable side effects of using the instrument: Once we automate the complexity calculation, it will not require considerable additional workload of manpower of the company. The only cost will be due to automation.

4.3.5.2 Evaluation of *SE* by Weyuker's Properties

As we did for our *C(XSD)* metric the *SE* metric has also been evaluated against the Weyuker's properties (see section 3.2) for establishing itself as a good and comprehensive measure and also to determine its strength and weaknesses.

Property 1: (∃P) (∃Q) (|P |≠| Q|). Where P and Q are programme body. As can be observed from Table 4.4 it is obvious that for different Schema documents different *SE* values are evaluated, hence this property is hold by *SE* metric.

Property 2: Let *c* be a non-negative number then there are only finitely many programs of complexity *c*. This property emphasis the non-coarseness of a software product: it constraints property one by stating that only a finite number of programs can be assigned the same complexity value c. All Schema documents consist of only finite number of elements. Now, as the *SE* measure depends upon the structures of Schema components and its calculation considers the total number of elements and the frequencies of them. Thus given that a Schema contains only finitely many elements, there are only finitely many Schemas being equal to the measure *c*. Hence, *SE* metric holds this property.

Property 3: There are distinct programs P and Q such that |P| = |Q|.
As pointed out by Weyuker, this property is not satisfied by only the measures which assigns a unique numerical name to each program and treats this name as a programs complexity. Therefore, it is clear that this property is hold by *SE*.

Property 4: (∃P) (∃Q) (P≡Q & | P | ≠ | Q |)
This property states that even though two programs compute the same function, their program complexities depend upon the implementation details. Consider the Schema document books.xsd and its modified version books2.xsd (see listing 4.6 and listing 4.10). In section 4.3.4 we showed that even though both Schema documents can confirm the same resulting XML document, the *SE* values for them are found different. Thus, *SE* metric does adhere to this property.

Property 5: (∀P) (∀Q) (|P| ≤ |P; Q| and |Q| ≤ |P; Q|).
This property states that the complexity values of two Schema documents P, Q should be less than or equal to the complexity of the composition of the two Schema documents. Consider the Schema documents books.xsd and projects.xsd. If we combine these two XSDs the resulting equivalence classes are:

C_1= {books,projects},

C_2= {book},

C_3= {title, firstname, lastname, firstname,
lastname,language},
C_4 = {author},
C_5 = {publisher}
C_6= { singletions, namespaces, scripts },
C_7= {project},
C_8= {singletion, namespace},
C_9 = {script}

The *SE* value for this combined Schema document is found:

$$SE_{combined.xsd} = -\sum_{i=1}^{9} P(C_i) log_2 P(C_i)$$
$$= -[(2/18)*\log_2(2/18) + (1/18)*\log_2(1/18) + (6/18)*\log_2(6/18) + (1/18)*\log_2(1/18) + (1/18)*\log_2(1/18) + (3/18)*\log_2(3/18) + (1/18)*\log_2(1/18) + (2/18)*\log_2(2/18) + (1/18)*\log_2(1/18)]$$
$$= 2.82189$$

In section 4.3.4 we have found $SE_{books.xsd}$ = 1.87996 and $SE_{projects.xsd}$ = 2.41938. Hence, since $SE_{books.xsd} < SE_{combined.xsd}$ and $SE_{projects.xsd} < SE_{combined.xsd}$ this property is satisfied by *SE*.

Property 6: (∃P) (∃Q) (∃R) (|P|=|Q|) & (|P; R|≠ |Q; R|)
This property asserts that we can find two Schema of equal *SE* value which when separately concatenated to a same third Schema yields the Schemas of different *SE* values. Assume we have three schema documents; P, Q and R that do not have elements in common i.e. their component definitions for their namespaces [33] are different. The first Schema document P.xsd having 4 equivalence classes consist of similar structured element and the number of elements in each class are C_1=3, C_2=3, C_3=1, C_4=1. The second Schema Q.xsd has elements completely different in structure from the elements of P.xsd and its equivalence classes with the number of their member elements are C_5=3,C_6=3,C_7=1,C_8=1.The last Schema R.xsd has similar elements as P.xsd has, that is it has the same structured elements and hence same graph representation as P.xsd has and its classes are: C_1=3, C_2=3, C_3=1, C_4=1.Note that Q and R do not have any element in common. We can find that the *SE* values for these three documents are equal and found as:

$$SE_{P,Q,R} = -\sum_{i=1}^{4} P(C_i)\log_2 P(C_i)$$
$$= -[(3/8)*\log_2(3/8) + (3/8)*\log_2(3/8) + (1/8)*\log_2(1/8) + (1/8)*\log_2(1/8)]$$
$$= 1.81128$$

If the Schema documents P and R are combined then the combined Schema has four classes since these two Schemas have the same graph representation with no common nodes. Hence the number of resulting equivalence classes with their member elements will be C_1=6, C_2=6, C_3=2, C_4=2 and:

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

If the Schemas Q and R are combined then the number of resulting classes will be eight since these Schemas also have fully different structured elements. The classes and member elements' counts of combination of Q, R will be C_1=3, C_2=3, C_3=1, C_4=1, C_5=3,C_6=3,C_7=1,C_8=1 and:

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

Since $SE_{(P;R)} \neq SE_{(Q;R)}$ this property is satisfied by our SE metric.

Property 7:

There are program bodies P and Q such that Q is formed by permuting the order of the statement of P and ($|P| \neq |Q|$).

In section 4.3.4 we modified books.xsd by using reference to global elements group to define its firstname and lastname elements for the author and the publisher elements, that is, we changed the place of the declaration/definition of its components. What we have found in section 4.3.4 is that the modified version books2.xsd has different *SE* value ($SE_{books2.xsd}$=2.80735) from its original version books.xsd ($SE_{books.xsd}$=1.87996) although both XSDs can confirm the same XML document. Therefore, this property is satisfied by *SE* metric.

Property 8: The value of *SE* is real numbers so renaming of XSD cannot change the value of *SE* metric. As a consequence, our *SE* metric clearly does adhere to this property.

Property 9: $(\exists P)(\exists Q)(|P|+|Q|) < (|P; Q|)$. By using the same example given under *property 5* this property is not satisfied by *SE* metric since we have found that:

$SE_{books.xsd} + SE_{projects.xsd} > SE_{combined.xsd}$. Since 1.87996 + 2.41938 > 2.82189

In this section we have validated our measures through Weyuker's properties and found that the proposed measures satisfy eight Weyuker's property out of nine, which established *SE* measure as well structured one. It is worth mentioned that a complexity measure are not supposed to satisfy all Weyuker's properties.

4.3.5.3 Empirical Validation of *SE*

The statistics we have collected after analyzing Schemas for empirical validation of our *SE* metric are shown in Table 4.4. We also calculate the values of *#E*, and *Fanning* metrics for analyzed Schemas to compare them with *SE* metric. Note that the data that resides in the fifth column is measures of *DSERS* metric which will be introduced in section 4.4. The graphs depicted in Figure 4.5 and 4.6 show the comparison results between *#E* and *SE* metrics and, between *Fanning* and *SE* respectively. We have assigned each Schema document with id numbers and the references to web links for these documents are shown in Table 2 in Appendix C.

It can be observed from Table 4.4 and Figure 4.5 the Schema documents that have equal *#E* metric values can be differentiated by *SE* metric in terms of their complexities. From Figure 4.5 one can easily see that it is not always the case increasing #E value result in increasing complexity of XSD measured by *SE* metric. Further XSDs (for example Schemas with id numbers 6, 7; 8, 9 and 14,15) having the same *#E* values do not have the same SE metric values. This is due to the fact that *#E* metric does not consider internal structures of XSDs and occurrences of these structures. Therefore it fails to capture psychological complexity of XSD due to frequencies of its similar structured elements.

Table 4.4 The metrics values of analyzed Schema documents for empirical validation of *SE* and *DSERS* metrics.

ID	#E	SE	FANNING	DSERS	Reference
1	8	1.299	0.750	3.750	1
2	8	1.436	0.375	2.375	2
3	12	1.252	0.667	6.000	4
4	17	1.902	0.588	4.882	71
5	17	2.278	0.765	4.765	72
6	18	1.352	0.556	7.778	13
7	18	1.723	0.556	6.667	20
8	19	1.578	0.579	7.421	14
9	19	2.715	0.579	3.316	28
10	20	1.522	0.400	7.200	11
11	22	1.730	0.636	9.636	9
12	25	1.940	0.720	6.600	73
13	29	1.760	0.586	11.414	74
14	30	1.273	0.533	13.467	75
15	30	1.711	0.533	10.867	26
16	37	2.667	1.000	8.027	48
17	37	1.363	0.730	20.676	16
18	37	1.724	0.757	9.973	35
19	41	2.160	0.512	11.098	52
20	42	1.462	0.881	23.476	15
21	43	2.700	0.953	7.744	44
22	44	2.289	0.727	11.591	46
23	45	1.267	0.933	29.089	10
24	45	3.082	0.844	7.267	50
25	46	2.189	1.370	17.174	76
26	47	1.742	0.660	20.957	77
27	48	2.123	0.917	13.417	78
28	52	1.513	0.846	29.885	37
29	54	1.769	0.630	17.556	79
30	60	1.919	0.800	26.500	47
31	60	2.579	1.083	14.400	38
32	63	2.634	1.159	14.238	53
33	65	2.693	1.077	17.677	57
34	68	1.858	0.882	32.206	39
35	83	1.873	0.867	35.434	80
36	83	1.474	0.590	36.084	64
37	84	2.845	1.536	13.905	65
38	84	3.053	1.369	15.048	63
39	111	2.936	1.820	27.036	81
40	153	1.447	0.902	93.732	82

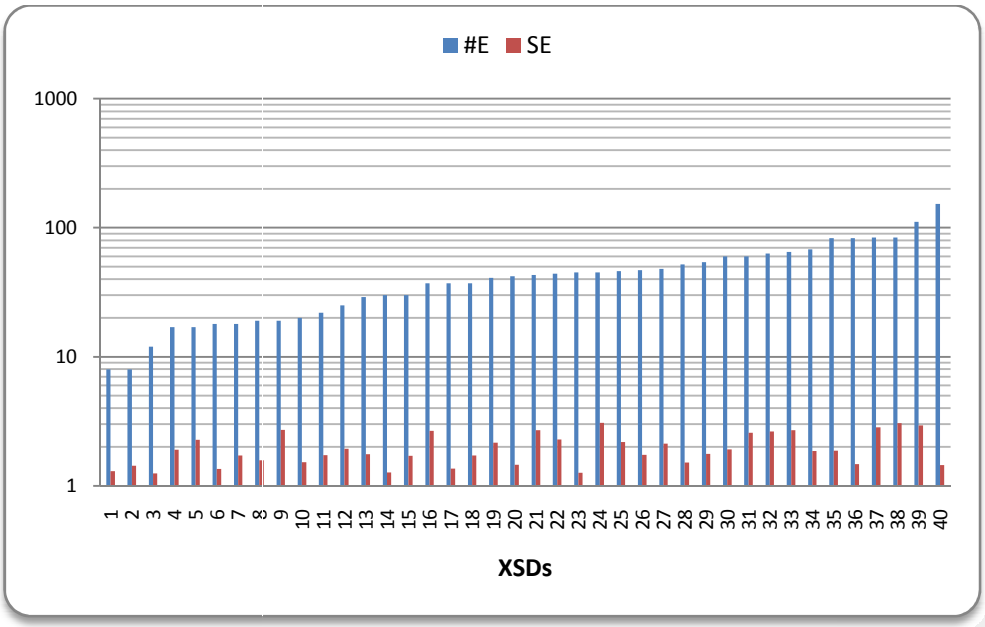

Figure 4.5 *SE* vs. #E metrics. The data is ordered by #E measures.

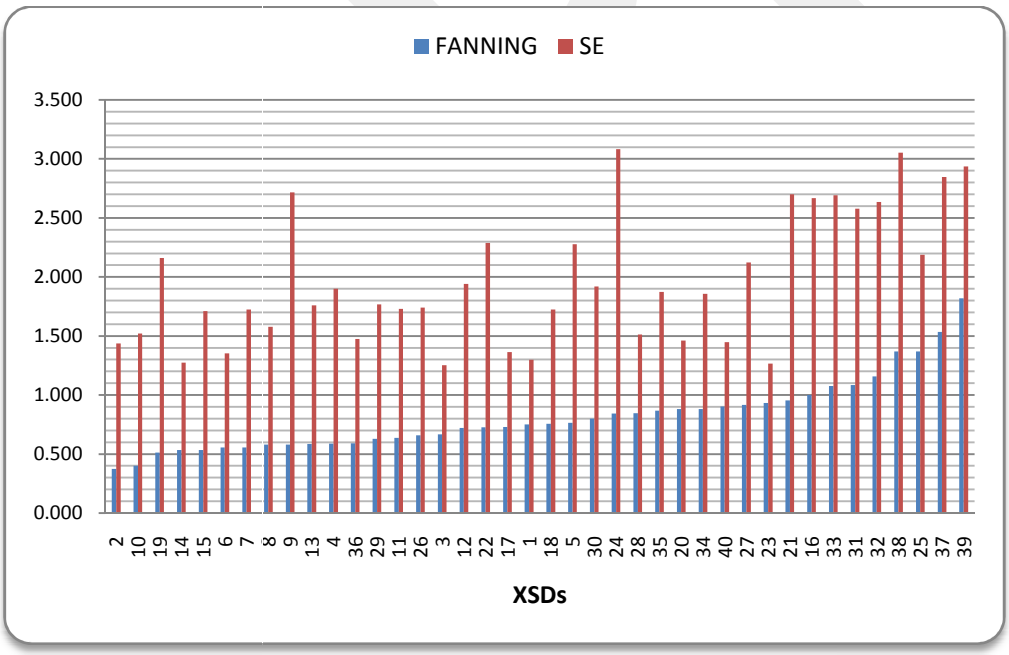

Figure 4.6 Element fanning vs. *SE metrics*. The data is ordered by *element fanning* measures.

On the other hand, while some of the *element fanning* measures given in Table 4.4 and depicted in Figure 4.5 are equal or very close to each other and increase very smoothly, *SE* metric does not exhibit the same behaviour.

The directed graph representations of those Schemas that have lower *SE* value tend to exhibit more regularity due to higher frequencies of similar structured elements.

Thus these XSDs are easy to comprehend since the developer gains familiarity which results in lower SE metric value. Like *#E* metric, this point is also missed by *element fanning* metrics.

4.3.6 Concluding Remark on *SE* Metric

We have presented the complexity metric based on the entropy concept from information theory [52] for measuring the physiological complexity of a given Schema document due to the diversity in its elements' structure. It was found that measuring the complexity of the Schema documents by Schema Entropy metric, *SE*, can provide a useful feedback when comparing Schemas documents that have equal number of elements. The directed graph representation of Schema documents that have more similar structured elements with higher frequencies of occurrences exhibit more regularity, thus are easy to grasp because of gained familiarity. An analogy with this situation can also be observed when one tries to investigate a binary tree and an irregular tree. It is obvious that understanding the structure and the relation between nodes of a binary tree is easier than that of irregular tree.

The *SE* metric can also be used to reflect the reusability of Schema components, which is one of our future works. One may also use entropy concept to measure diversity in type definitions in Schema documents, in order to compare Schemas that have equal number of complex type definitions.

In the validation criteria, we used the Weyuker's properties and found that that the *SE* measure satisfies eight Weyuker's property out of nine, which established this measure as well, structured one and evaluation against Kaner's framework proves its practical applicability. Further, the empirical validation of the *SE* metric also verifies its usefulness to measure physiological complexities of Schema documents that have equal number of element definitions. Hence, we can conclude that in terms of their complexity values, XSDs having equal *#E* can easily be differentiated by using *SE* metric.

4.4 Distinct Structured Elements Repetition Scale Metric (*DSERS*)

Distinct Structured Repetition Scale (*DSERS*) metric is similar to the *SE* metric and is intended to analyze variety in structures of XSD elements. It is adapted from ARS

metric which is proposed by Boxall [43] 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 (Argument Repetition Scale)* 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 specialized 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 arguments declarations. With this example he drew a conclusion that more consistent argument declarations allow better knowledge transferability and make it easier to understand the interfaces.

4.4.1 Motivation

As stated earlier when we encountered similar structured elements appearing with high frequencies of occurrences comprehending a Schema document became easier due to gained familiarity since the directed graph of such a Schema exhibits more regularity. Knowing the overall structure of a graph representation of a Schema whose nodes are connected to the same number of other nodes and possess regularity makes understanding the data dependency between these nodes much easier in contrast to non-regular structured graph. In this aspect, when a comparison is needed between two graphs both having equal number of nodes it is expected that regular structured graph is easier to understand. By considering the regularity in elements' structures of XSD we developed *DSERS* metric to capture the affects of frequency of occurrences of similar structured elements on the physiological complexity of Schema document.

4.4.2 Definition of *DSERS* Metric

Distinct Structured Repetition Scale can be defined as:

$$\text{DSERS} = \frac{\sum_{i=1}^{p} de_i^{\,2}}{\#e}$$

where p is number of equivalence classes, de_i is the number of members inside the i^{th} class and #e is the total number of element nodes in the graph representation of XSD. The possible values for *DSERS* will be in the range $1 \leq DSERS \leq \#e$. The higher *DSERS* indicates lower physiological complexity of XSD. In other words *DSERS* metric has inverse relation with Schema complexity.

The relation between *DSERS* and *SE* is that higher *DSERS* and lower *SE* indicate lower physiological complexity of XSD. This is because while the calculation of *DSERS* is based on squaring the frequencies of similar structured elements *SE* calculation is based on logarithmic scale with base two. Hence, higher *DSERS* and lower *SE* can be interpreted that the developer makes less effort to understand and remember the structures of elements and dependency between them due to gained familiarity with elements having higher frequency of occurrences.

4.4.3 Illustration of *DSERS* Metric

In order to demonstrate the *DSERS* metric we use the same example Schema documents that are given in listing 4.6, 4.7 and 4.10 respectively and, their directed graph representations that are shown in Figure 4.2, 4.3 and 4.4.

Example 4.4: DSERS values for the Schemas books.xsd, projects.xsd and books2.xsd are calculated by considering their corresponding equivalence classes given in listing 4.8, 4.9 and 4.11. The counts of distinct structured elements appearing in the graph representations of these three Schemas are equal to the number of their equivalence classes and are found as 5, 6, and 6 respectively. Hence,

$$DSERS_{books.xsd} = \frac{\sum_{i=1}^{5} de_i^{\,2}}{9}$$
$$= (1^2+1^2+5^2+1^2+1^2)/9$$
$$= 29/9$$
$$= 3.222$$

$$DSERS_{projects.xsd} = \frac{\sum_{i=1}^{6} de_i^{\,2}}{9}$$
$$= (1^2+1^2+3^2+2^2+1^2+1^2)/9$$
$$= 17/9$$
$$= 1.889$$

$$DSERS_{books2.xsd} = \frac{\sum_{i=1}^{6} de_i{}^2}{7}$$
$$= (1^2+1^2+2^2+1^2+1^2+1^2)/7$$
$$= 9/7$$
$$= 1.286$$

As can be seen from example 4.4 although books.xsd and projects.xsd have equal number of elements, the Schema projects.xsd has six different structured elements i.e. variety in its element structures is higher thus its graph is more irregular. Since it is harder to remember and comprehend a Schema document as variety in its elements' structures increases comprehending books.xsd is easier than projects.xsd. On the other hand, although books.xsd and books2.xsd conforms the same XML instance and books.xsd has more elements *DSERS* value for books.xsd is less than that of books2.xsd. This is due to the fact; the graph of Schema books.xsd consists of less number of distinct structured elements and similar elements appear with higher frequencies (resulting in more regularity in its graph).

The *DSERS* values and other Schema measures for these example documents are given in Table 4.5. As can be observed from Table 4.5, although #E, fanning, *C(XSD)* metrics values are consistent to each other for books.xsd and projects.xsd documents as *SE* metrics does *DSERS* metrics evaluates different measure for them. The values of *SE* and *DSERS* metrics are also consistent to each other, since lower *SE* and higher *DSERS* indicates less physiological complexity of Schema due to less variety in element structures. On the other hand, when books.xsd and books2.xsd are compared according to *DSERS* measure books2.xsd posses more diversity in elements' structures and less frequency of similar structured elements.

Table 4.5 The values of complexity measures developed for XSDs.

Schema	*Listing*	*Graph Fig. No*	*Fanning*	*#E*	*C(XSD)*	*SE*	*DSERS*
books.xsd	*4.4*	*4.2*	*8/9*	*9*	*12*	*1.87996*	*3.222*
projects.xsd	*4.5*	*4.3*	*8/9*	*9*	*12*	*2.41938*	*1.889*
books2.xsd	*4.9*	*4.4*	*8/7*	*7*	*12*	*2.80735*	*1.286*

The Schema documents with higher *DSERS* tend to be dominated by fewer distinct structured elements that have higher frequency of occurrences. As larger XSDs in terms of elements' count generally have more elements, *DSERS* measure may tend to

be higher. However, this reflects the notion that larger XSDs are able to contain more repetition, and so *DSERS* reflects this. It is likely that real world XSDs will not have *DSERS* measurements approaching upper limit which is the total number of elements, #e. Hence, like *SE* metric the *DSERS* metric can be useful in comparing XSDs having equal number of elements.

4.4.4 Validation of *DSERS* Metric

In Section 4.4.4.1 the applicability of *DSERS* is checked through theoretical and empirical validations and it is examined against the practical framework developed by Kaner [69]. We theoretically evaluated our metric against Weyuker's [73] set of measurement principle in section 4.4.4.2. The empirical validation of *DSERS* metric is given in section 4.4.4.3.

4.4.4.1 Practical Evaluations of *DSERS* Metric

The difference between *SE* metric and *DSERS* metric is their calculation method only, hence lower *SE* and higher *DSERS* indicates less physiological complexity of Schema due to less variety in element structures. Both metrics consider variety in element structures of XSD and occurrence frequencies of them. Thus, for XSDs having equal number of elements both measures indicate the same result about complexity levels of XSDs. Since both *SE* and *DSERS* metrics examine Schema documents in the same concept we do not repeat practical evaluation of *DSERS* metric here. However, for the r*elationship between the attribute to the metric value* there is a direct relation between the quality of the Schema and our metric *DSERS*. If the *DSERS* value increases, it is clear that the Schema quality will increase. This is because higher *DSERS* can be interpreted that the developer makes less effort to understand and remember the structures of elements and dependency between them due to gained familiarity with elements having higher frequency of occurrences.

4.4.4.2 Evaluation of *DSERS* by Weyuker's Properties

In the following paragraphs our metric *DSERS* has been evaluated against the Weyuker's properties for establishing itself as a good and comprehensive measure and also to determine its strength and weaknesses.

Property 1: (∃P) (∃Q) (|P| ≠ |Q|). Where P and Q are programme body. This property is satisfied since it is obvious from Table 4.4 that for different Schema documents different *DSERS* values can be evaluated; hence *DSERS* metric does adhere to this property.

Property 2: Let *c* be a non-negative number then there are only finitely many programs of complexity *c*. All Schema documents consist of only finite number of elements. Now, like SE metric the *DSERS* metric depends upon the structures of Schema components and its calculation considers the total number of elements and the frequencies of them. Thus given that a Schema contains only finitely many elements, there are only finitely many Schemas being equal to the measure *c*. Hence, *DSERS* metric holds this property.

Property 3: There are distinct programs P and Q such that |P| = |Q|.
Since DSERS metric does not assign a unique numerical name to each Schema file and does not treat this name as a Schema complexity it is clear that *DSERS* metric does adhere to this property.

Property 4: (∃P) (∃Q) (P≡Q & | P | ≠ | Q |)
Consider the Schema document books.xsd and its modified version books2.xsd. In section 4.4.3 we found that even though both Schema documents can confirm the same resulting XML document, the *DSERS* values ($DSERS_{books.xsd} = 3.222$ and $DSERS_{books2.xsd} = 1.286$) for them are found different. Thus, *DSERS* metric does adhere to this property.

Property 5: (∀P) (∀Q) (|P| ≤ |P; Q| and |Q| ≤ |P; Q|).
Assume we have two Schema documents P.xsd and Q.xsd .The first Schema document P.xsd having 4 equivalence classes consist of similar structured element and the number of elements in each class are C_1=3, C_2=3, C_3=1, C_4=1. The second Schema Q.xsd has similar elements as P.xsd has, that is it has the same structured elements and hence same graph representation as P.xsd has and its classes are*:* C_1=3, C_2=3, C_3=1, C_4=1.We can find that the *DSERS* values for P and Q are equal and found as:

$$DSERS_P = DSERS_Q = \frac{\sum_{i=1}^{4} de_i^2}{8}$$

$$= (3^2+3^2+1^2+1^2)/8$$
$$= 2.5$$

If the Schema documents P and Q are combined then the combined Schema has four classes since these two Schemas have the same graph representation with no common nodes. Hence the number of resulting equivalence classes with their member elements will be C_1=6, C_2=6, C_3=2, C_4=2 and:

$$DSERS_{(P;Q)} = \frac{\sum_{i=1}^{4} de_i^{\,2}}{16}$$
$$= (6^2+6^2+2^2+2^2)/16$$
$$= 5$$

We have found $DSERS_{P.xsd} = 2.5$, $DSERS_{Q.xsd} = 2.5$ and $DSERS_{(P;Q)} = 5$. Hence, since $DSERS_{P.xsd} < DSERS_{(P;Q)}$ and $DSERS_Q < DSERS_{(P;Q)}$this property is satisfied by *DSERS*.

Property 6: $(\exists P)\ (\exists Q)\ (\exists R)\ (|P|=|Q|)\ \&\ (|P; R| \neq |Q; R|)$

Assume we have three Schema documents; P, Q and R that do not have elements in common. The first Schema document P.xsd having 4 equivalence classes consist of similar structured element and the number of elements in each class are C_1=3, C_2=3, C_3=1, C_4=1. The second Schema Q.xsd has elements completely different in structure from the elements of P.xsd and its equivalence classes with the number of their member elements are C_5=3,C_6=3,C_7=1,C_8=1.The last Schema R.xsd has similar elements as P.xsd has, that is it has the same structured elements and hence same graph representation as P.xsd has and its classes are: C_1=3, C_2=3, C_3=1, C_4=1.Note that Q and R do not have any element in common. We can find that the *DSERS* values for these three Schema documents are equal and found as:

$$DSERS_P = DSERS_Q$$
$$= DSERS_R = \frac{\sum_{i=1}^{4} de_i^{\,2}}{8}$$
$$= (3^2+3^2+1^2+1^2)/8$$
$$= 2.5$$

If the Schema documents P and R are combined then the combined Schema has four classes since these two schemas have the same graph representation with no common

nodes. Hence the number of resulting equivalence classes with their member elements will *be* C_1=6, C_2=6, C_3=2, C_4=2 and:

$$DSERS_{(P;R)} = \frac{\sum_{i=1}^{4} de_i^{\,2}}{16}$$
$$= (6^2+6^2+2^2+2^2)/16$$
$$= 5$$

If the Schemas Q and R are combined then the number of resulting classes will be eight since these Schemas also have fully different structured elements. The classes and member elements' counts of combination of Q, R will be C_1=3, C_2=3, C_3=1, C_4=1, C_5=3,C_6=3,C_7=1,C_8=1 and:

$$DSERS_{(Q;R)} = \frac{\sum_{i=1}^{8} de_i^{\,2}}{16}$$
$$= (3^2+3^2+1^2+1^2+3^2+3^2+1^2+1^2)/16$$
$$= 2.5$$

Since $DSERS_{(P;R)} \neq DSERS_{(Q;R)}$ this property is satisfied by our *DSERS* metric.

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

What we have found in section 4.4.3 is that the modified version books2.xsd has different *DSERS* value ($DSERS_{books2.xsd}$=1.286) from its original version books.xsd ($DSERS_{books.xsd}$=3.222) although both XSDs can confirm the same XML document. Therefore, this property is satisfied by *DSERS* metric.

Property 8: The value of *DSERS* is real numbers so renaming of XSD cannot change the value of *DSERS* metric. As a consequence, our *DSERS* metric clearly does adhere to this property.

Property 9: $(\exists P)\ (\exists Q)\ (|P|+|Q|) < (|P;Q|)$.
By using the same example given under *property 5* this property is not satisfied by DSERS metric since we have found that:
$DSERS_{P.xsd}$+ $DSERS_{Q.xsd}$>$DSERSE_{(P;Q)}$. Since 2.5+ 2.5 = 5

In this section we have validated our measures through Weyuker's properties and found that the proposed measures satisfy eight Weyuker's property out of nine, which established *DSERS* measure as well structured one.

4.4.4.3 Empirical Validation of *DSERS*

For empirical validation of *DSERS* metric we have analyzed the same Schema documents that have been used to evaluate *SE* measures. The statistics we have collected after analyzing these Schemas to evaluate *DSERS* measures are shown in Table 4.4. The graphs depicted in Figure 4.7 and 4.8 show the comparison results between *#E* and *DSERS* metrics and, between *Fanning* and *DSERS* respectively. It can be observed from Table 4.4 and Figure 4.7 the Schema documents that have equal *#E* metric values can also be differentiated by *DSERS* metric in terms of their complexities. However, the graphs of those Schemas that have higher *DSERS* measure tend to exhibit more regularity due to higher frequencies of similar structured elements, thus they are easy to comprehend since the developer gains familiarity.

On the other hand while some of the *element fanning* measures in Table 4.4 is equal or very close to each other for the Schema documents (for example Schemas with id numbers 6, 7; 8, 9 and 14,15) the *DSERS* metric evaluates different values for them. The graph depicted in Figure 4.8 shows the comparison result between element Fanning and *DSERS* metrics.

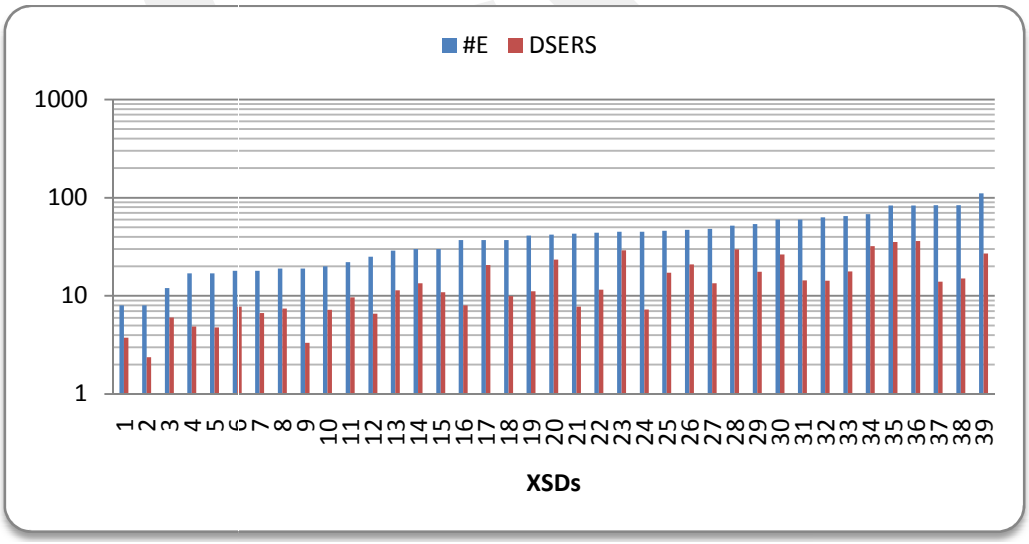

Figure 4.7 *#E* metric vs. *DSERS* metric. The data is ordered by #E measures.

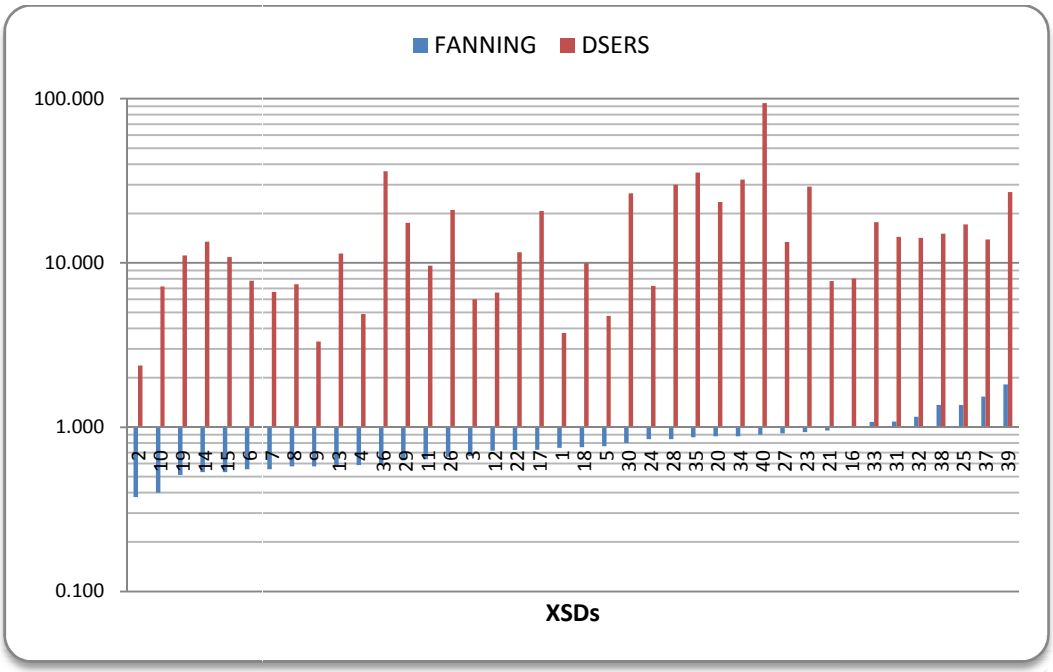

Figure 4.8 Element fanning vs. DSERS. The data is ordered by *element fanning* measure.

4.4.5 Concluding Remark on *DSERS* Metric

We have presented a new metric *DSERS* based on the occurrences of similar structured elements of Schema documents for measuring the physiological complexity of XSD due to the diversity in its elements' structure. It was found that measuring the complexity of the XSDs by *DSERS* can provide a useful feedback when comparing Schemas that have equal number of elements. The graph representations of Schema documents that have more similar structured elements with higher frequencies of occurrences exhibit more regularity, thus are easy to grasp because of gained familiarity.

The *DSERS* metric is validated both empirically and theoretically. By analyzing real life XSDs the validity and usefulness of the *DSERS* metric to measure physiological complexities of Schema documents that have equal number of element definitions is verified. Theoretical evaluation of *DSERS* against Weyuker's properties shows that *DSERS* measure is established as a well structured one and evaluation against Kaner's framework proves its practical applicability. Hence, in terms of their complexity values, XSDs having equal *#E* can easily be differentiated by using *DSERS* metric.

CHAPTER 5

W3C DOCUMENT TYPE DEFINITION (DTD) METRICS

5.1 Introduction

In this chapter in order to measure the complexity of schema documents that are written in W3C Document Type Definition (DTD) [37] language we presented a new metrics by adopting our *SE* and *DSERS* introduced in section 4.3 and 4.4 metrics to DTDs. In section 5.2 existing metrics for DTD is presented. Our motivation to adopt entropy metric to DTD is explained in section 5.3. The definitions of our DTD metrics*, E (DTD), DSERS (DTD)* are given in section 5.4. Illustration of these metrics and validations of them are given in section 5.5 and section 5.6 respectively.

5.2 Existing Metrics for DTD

Arnaud Sahuguet [13] analyzed the typical characteristics of DTDs and presented count based measures such as number of elements, attributes, ID and IDREF entities etc. Choi *et al*. [12, 14] discussed some criteria about how DTDs should be designed for ensuring the quality of XML projects and pointed out the challenges in XML design due to DTDs characteristics. In [19] some well known metrics that were originally developed for traditional software products, such as LOC, McCabe, Fan-in and Fan-out, Depth of Inheritance Tree (DIT) were adapted to DTDs for measuring complexity of DTDs. *Mustafa et al.* [15] demonstrated that the XML documents that are generated by the DTD with higher nesting levels have higher weights and more complicated compared to the documents with lower nesting levels. In this demonstration various techniques were used to represent XML documents as a regular expression and by determining complexity values of regular expression; a tree representation of XML documents and the implementation of Weight Allocation (WA) algorithm.

5.3 Motivation

The common approach in most of the proposed metric for schema documents written in DTD is that none of them consider the structural similarities in the DTD components. However, as we stated in section 4.3 similarity in the schema components' structure affects the effort required for comprehending the schema document. The more similar structured elements and attributes the schema document has the easier to comprehend and remember the schema document is.

5.4 Definitions of *E(DTD)* and *DSERS(DTD)* Metric

The entropy concept was explained in section 4.3 when we introduced *SE* metric, hence we do not repeat the same concept here. As we did for defining *SE* metric in section 4.3.3, for the assessment of structural complexity of a given DTDs the elements appearing in the corresponding directed graph are grouped in to the equivalence classes based on their *fan-in, fan-out* and number of attributes. That is, elements that have the same value of *fan-in* and *fan-out* and number of attributes are interpreted as elements having the same structural complexity. By using the below equation,

$$\begin{aligned} E(DTD) &= -\sum_{i=1}^{n} P(C_i)\log_2 P(C_i) \\ &= -\sum_{i=1}^{n} (1/n)\ \log_2(1/n) \\ &= \log_2 n \end{aligned}$$

where n is the number of equivalence classes.

By following similar approach taken for evaluating *DSERS* metric introduced in section 4.4 we can use the same formula for the formal definition of *DSERS (DTD)* metric:

$$DSERS(DTD) = \frac{\sum_{i=1}^{p} de_i^{\,2}}{\#e}$$

where p is number of equivalence classes, de_i is the number of members inside the i^{th} class and $\#e$ is the total number of element nodes in the graph representation of DTD.

5.5 Illustration of *E(DTD)* and *DSERS(DTD)* Metrics

We have demonstrated our *E(DTD)* metric by three example schema documents: *publications.dtd, order.dtd and tvschedule.dtd s*hown in listing 5.1, listing 5.2 and listing 5.3 respectively. The directed graph representations of the three DTDs are given in Figure 5.1, Figure 5.2, and Figure 5.3. From the directed graph representations of these three DTDs we evaluated the equivalence classes of them that are given listing 5.4, listing 5.5 and listing 5.6 respectively.

```
<!ELEMENT publications
(book | article | conference)*>
<!ELEMENT book
(title, author+, edition, publisher)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (Firstname, Lastname)>
<!ELEMENT Firstname (#PCDATA)>
<!ELEMENT Lastname (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT article ((#PCDATA)>
<!ELEMENT conference
(editor+, conftitle, city, year)>
<!ELEMENT conftitle (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

Listing 5.1 The schema document publications.dtd.

```
<!ELEMENT Order (Customer, Part+)>
<!ATTLIST  id CDATA #REQUIRED>
<!ELEMENT Customer (Firstname, Lastname, Email)>
<!ELEMENT Firstname (#PCDATA)>
<!ELEMENT Lastname (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Part (key, Quantity, ExtendedPrice, Tax, Shipment+)>
<!ELEMENT key (#PCDATA)>
<!ELEMENT Quantity (#PCDATA)>
<!ELEMENT ExtendedPrice (#PCDATA)>
<!ELEMENT Tax (#PCDATA)>
<!ATTLIST Part color CDATA #REQUIRED>
<!ELEMENT Shipment (ShipDate, ShipMode, Address)>
<!ELEMENT ShipDate (#PCDATA)>
<!ELEMENT ShipMode (#PCDATA)>
<!ELEMENT Address(#PCDATA)>
```

Listing 5.2 The schema document Order.dtd.

```
<!ELEMENT TvSchedule (Name,Channel+)>
<!ELEMENT Channel (Banner,Day+)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Banner (#PCDATA)>
<!ELEMENT Day(Date,(Holiday|ProgramslotT+)+)>
<!ELEMENT Holiday (#PCDATA)>
<!ELEMENT Date(#PCDATA)>
<!ELEMENT Programslot (Time, Title, Description?)>
<!ELEMENT Time (#PCDATA)>
<!ELEMENT Title (Rating, Language)>
<!ELEMENT Rating (#PCDATA)>
<!ELEMENT Language (#PCDATA)>
 <!ELEMENT Description (#PCDATA)>
<!ATTLIST Channel Chan CDATA REQUIRED>
<!ATTLIST Programslot Vtr CDATA #IMPLIED>
```

Listing 5.3 The schema document tvschedule.dtd.

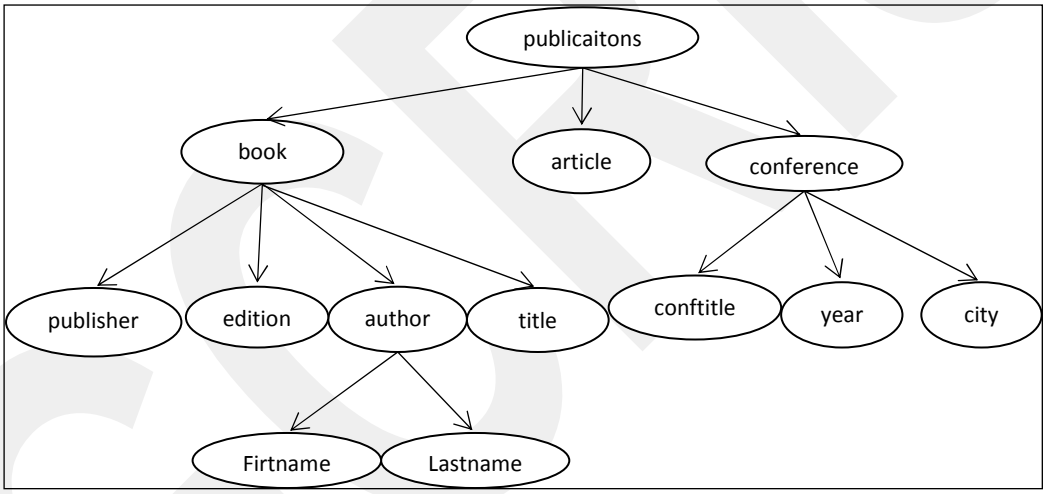

Figure 5.1 The directed graph representation of schema document publications.dtd.

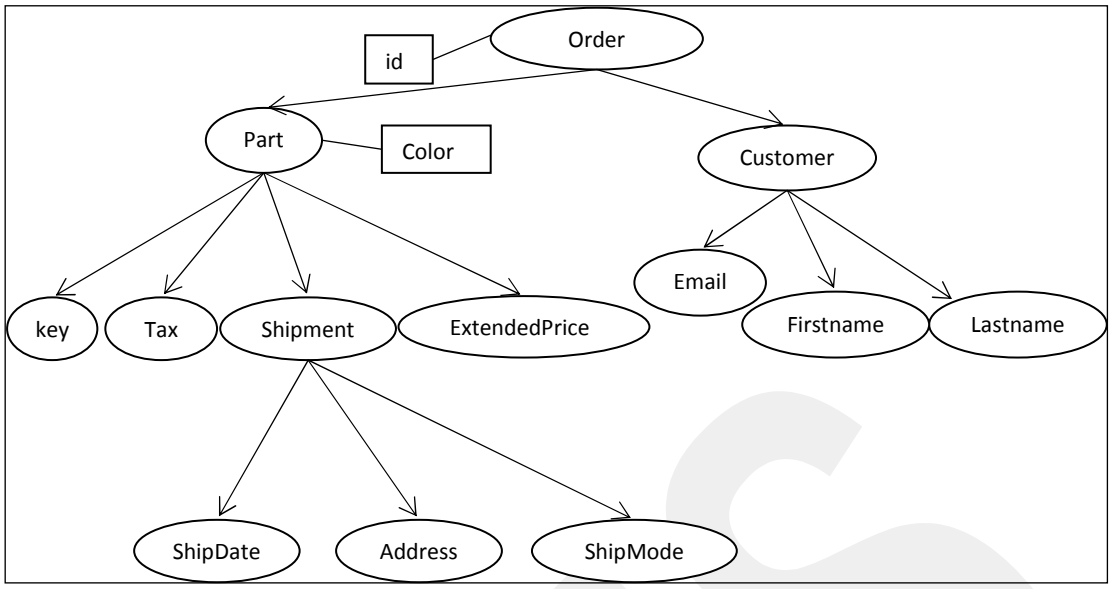

Figure 5.2 The directed graph representation of schema document Order.dtd.

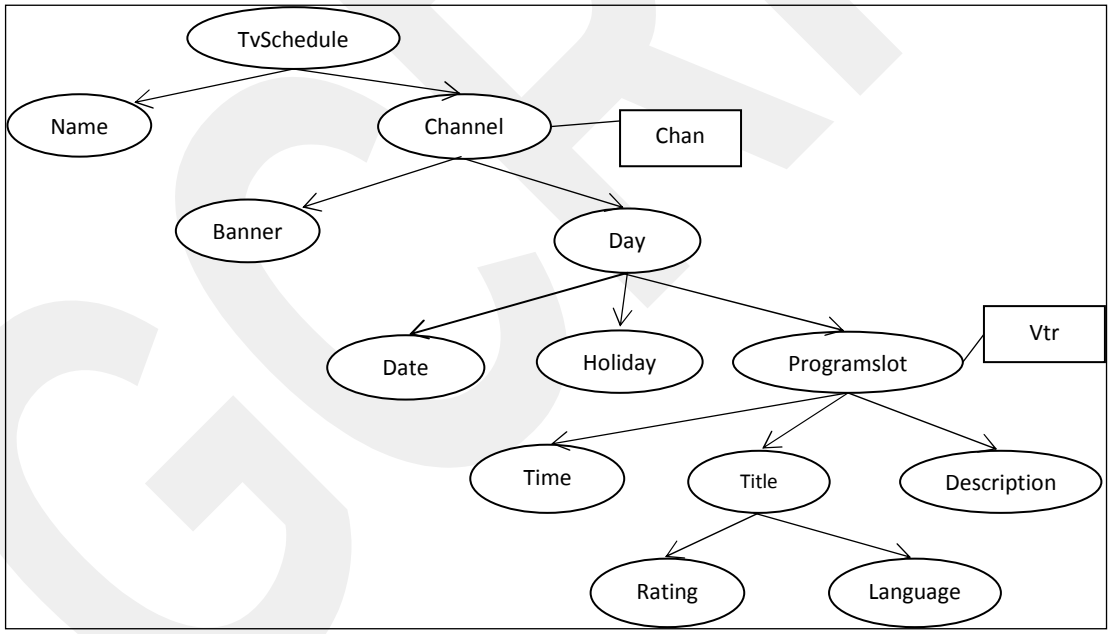

Figure 5.3 The directed graph representation of schema document tvschedule.dtd.

In order to obtain equivalence classes of these three example DTDs we grouped their elements according to the elements' *fan-in, fan-out* and number of attributes.

C_1= {publications},

C_2= {book},

C_3= {author},

C_4= {conference},

C_5 = {article, publisher, edition, title, conftitle, year, city, Firstname, Lastname}

Listing 5.4 Equivalence classes of the DTD publications.dtd

C_1= {Order},

C_2= {Part},

C_3= {Customer, Shipment},

C_4= {Key, Tax, ExtendedPrice, Email, Firstname, Lastname, Shipdate, Address, ShipMode}

Listing 5.5 Equivalence classes of the DTD Order.dtd:

C_1= {TvSchedule},

C_2= {Channel},

C_3= {Day},

C_4= {Programslot},

C_5= {Title},

C_6={Name, Banner, Language Date, Holiday, Time, Description, Rating }

Listing 5.6 Equivalence classes of the DTD tvschedule.dtd **.**

Having grouped the elements of each schema document into equivalence classes now we can calculate *E(DTD)* and *DSERS(DTD)* metrics values for them.

The *E(DTD)* and *DSERS(DTD)* values for the schema document given in listing 5.1 is:

$$E(DTD)_{publications.dtd} = -\sum_{i=1}^{5} P(C_i)\log_2 P(C_i)$$
$$= -[(1/13)*\log_2(1/13)+(1/13)*\log_2(1/13)$$
$$+(1/13)*\log_2(1/13)+(1/13)*\log_2(1/13)$$

$$+ (9/13)*\log_2(9/13)]$$
$$= 1.50588$$

$$DSERS(_{publications.dtd}) = \frac{\sum_{i=1}^{5} de_i^{\,2}}{\#e}$$
$$= (1^2+1^2+1^2+1^2+9^2)/13$$
$$= 85/13$$
$$= 6.53846$$

The *E(DTD)* and *DSERS(DTD)* values for the schema document given in listing 5.2 is:

$$E(D\ TD)_{Order.dtd} = -\sum_{i=1}^{4} P(C_i)\log_2 P(C_i)$$
$$= -[(1/13)*\log_2(1/13)+(1/13)*\log_2(1/13)$$
$$+(2/13)*\log_2(2/13) + (9/13)*\log_2(9/13)]$$
$$= 1.35203$$

$$DSERS(_{Order.dtd}) = \frac{\sum_{i=1}^{4} de_i^{\,2}}{\#e}$$
$$= (1^2+1^2+2^2+9^2)/13$$
$$= 87/13$$
$$= 6.69231$$

The *E(DTD)* and *DSERS(DTD)* values for the schema document given in listing 5.3 is:

$$E(DTD)_{tvschedule.dtd} = -\sum_{i=1}^{6} P(C_i)\log_2 P(C_i)$$
$$= -[(1/13)*\log_2(1/13)+(1/13)*\log_2(1/13)$$
$$+(1/13)*\log_2(1/13)+(1/13)*\log_2(1/13)$$
$$+(1/13)*\log_2(1/13)+ (8/13)*\log_2(8/13)]$$
$$= 1.85429$$

$$DSERS(_{tvschedule.dtd}) = \frac{\sum_{i=1}^{6} de_i^{\,2}}{\#e}$$
$$= (1^2+1^2+1^2+1^2+1^2+8^2)/13$$
$$= 69/13$$
$$= 5.30769$$

In order to compare the usefulness of *E(DTD)* and *DSERS(DTD)* metrics we also calculated the structural complexities of the three schema documents by using the element fanning [72] and tree impurity[4, 18] metrics and these values are shown in Table 5.1. In the third column the size metric [19] value that is evaluated by counting

the number of elements and attributes of the schema document for each DTD file is shown for comparison. The fourth and fifth columns give fanning and tree impurity metrics values. The values residing in the sixth and last column, *E(DTD)* and *DSERS(DTD)*, are calculated by considering the equivalence classes of the three DTDs.

The element fanning for a given DTD is:

$$\text{Fanning} = e/n$$

where ***e*** is the number of edges and ***n*** is the number of nodes in the directed graph of the schema. It can be interpreted that as the fanning value increases so is the complexity of a given schema document. The tree impurity of a given DTD can be calculated as:

$$\text{TI}=\frac{2(e-n+1)}{(n-1)(n-2)} * 100\%$$

where ***e*** is the number of edges and ***n*** is the number of nodes in the directed graph of the DTD. The tree impurity, (TI), metric was defined by Fenton *et al*. [4] and was used by Visser[18] as a structural complexity measure for the schema documents. The TI indicates the degree of deviation from a tree structure with the same number of nodes and the lower value is better. From Table 5.1 it can be observed that the values of structural complexity measures, the fanning, tree impurity and size metric are consistent to each others.

Table 5.1 The structural complexity values of the example DTDs.

DTD Listing No	***Graph Figure No***	***Size***	***Fanning***	***TI***	***E(DTD)***	***DSERS(DTD)***
5.1	*5.1*	*13*	*12/13*	*0%*	*1.50588*	*6.53846*
5.2	*5.2*	*13*	*12/13*	*0%*	*1.35203*	*6.69231*
5.3	*5.3*	*13*	*12/13*	*0%*	*1.85429*	*5.30769*

However, *E(DTD)* metric gives the different values for the structural complexities of the three schema documents. It is due to the fact; the *E(DTD)* metric considers the structural complexities of each element of DTD by grouping these elements into the same equivalence classes. That is the elements and attributes that have similar

structures and are grouped into the same equivalence classes can be interpreted as to have the same complexities. As the diversity in structures increases so does the number of equivalence classes which results in the more complex schema documents (since understanding and remembering the elements and attributes having different structure becomes more difficult).

The schema documents that exhibit less variety in structure of elements have the lower value of *E(DTD)* and higher value of *DSERS(DTD)* than the schema documents that exhibit greater variety in structure of elements. Hence, higher *DSERS(DTD)* and lower *E(DTD)* can be interpreted that the developer makes less effort to understand and remember the structures of elements and dependency between them due to gained familiarity with elements having higher frequency of occurrences. From Table 5.1 it can easily be observed that although, the three schema DTDs have equal values for the size, fanning and tree impurity metrics the complexities of them reflected by *E(DTD)* and *DSERS(DTD)* metrics are different. Therefore, the *E(DTD)* and *DSERS(DTD)* metrics are more realistic for measuring the structural complexity of a given schema than that of the others and in terms of the complexity it can be useful in differentiating and ranking DTDs that have equal number of elements.

5.6 Validation of *E(DTD)* and *DSERS(DTD)* Metrics

In order to check the applicability of *E(DTD)* and *DSERS(DTD)* metrics through theoretical validation we can examine them against the practical framework developed by Kaner[69] and evaluate them by using Weyuker's properties[73]. Since practical evaluation process was explained in section 4.3.5.1 and the development of *E(DTD)* and *DSERS(DTD)* metrics are based on the similar way taken in developing *SE* and *DSERS* metrics for XSDs we do not revisit practical evaluation here. For empirical validation of these two metrics we have analyzed 20 real DTD files downloaded from web. The statistics collected from these DTDs are shown in Table 5.2 and the graph based on these statistics is depicted in Figure 5.1. Analyzing much more DTD is aimed as one of our future works. In the following section we evaluate our DTD metrics through Weyuker's properties.

5.6.1 Evaluation of E (DTD) and DSERS (DTD) by Weyuker Properties

Since we adopted SE and DSERS metric to DTD it is clear that *Property 1, Property 2, and Property 3* are also satisfied by *E (DTD)* and *DSERS (DTD*) metrics.

In order for two DTD documents to generate the same XML documents both should be designed in the same structure since we do not consider number of occurrences of the elements in the resulting XML documents. That is, the implementation of DTDs should be the same for validation of the same XML document. Therefore property 4 cannot be satisfied by both *E(DTD)* and *DSERS(DTD)* metrics.

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

Consider example DTD documents given in listing 5.4 and listing 5.5. If we combine these two DTDs then the equivelance classes of the resulting DTD are:

C_1= {publications},

C_2= {book},

C_3= {author},

C_4= {conference, Customer, Shipment},

C_5 = {article, publisher, edition, title, conftitle, year, city, Key, Tax, ExtendedPrice, Email, Shipdate, Address, ShipMode}

C_6= {Order},

C_7= {Part},

C_8= {Firstname, Lastname}

Note that the elements Firstname and Lastname are common in both DTDs, hence the *fan-in, fan-out* and number of attributes of these two elements are 2, 0 and 0 respectively in the directed graph of combined DTD. The metrics values are:

$$E(DTD)_{publications.dtd;Order.dtd} = -\sum_{i=1}^{8} P(C_i)\log_2 P(C_i)$$

$$= -[(1/24)*\log_2(1/24) + (1/24)*\log_2(1/24) + (1/24)*\log_2(1/24) + (3/24)*\log_2(3/24) + (14/24)*\log_2(14/24) + (1/24)*\log_2(1/24) + (1/24)*\log_2(1/24) + (2/24)*\log_2(2/24)]$$

$$= 2.08255$$

$$\text{DSERS}(_{\text{publications.dtd;Order.dtd}}) = \frac{\sum_{i=1}^{8} de_i^{\,2}}{\#e}$$

$$= (1^2+1^2+1^2+3^2+14^2+1^2+1^2+2^2)/24$$

$$= 8.91667$$

In section 5.5 we found that

$$E(\text{DTD})_{\text{publications.dtd}} = 1.50588 \text{ and}$$

$$E(\text{DTD})_{\text{Order.dtd}} = 1.35203.$$

Since

$$E(\text{DTD})_{\text{publications.dtd}} < E(\text{DTD})_{\text{publications.dtd;Order.dtd}} \text{ and}$$

$$E(\text{DTD})_{\text{Order.dtd}} < E(\text{DTD})_{\text{publications.dtd;Order.dtd}}$$

our *E(DTD)* metric satisfies this property.

In section 5.5 we found that

$$\text{DSERS}(_{\text{publications.dtd}}) = 6.53846 \text{ and}$$

$$\text{DSERS}(_{\text{Order.dtd}}) = 6.69231.$$

Since

$$\text{DSERS}(_{\text{publications.dtd}}) < \text{DSERS}(_{\text{publications.dtd;Order.dtd}}) \text{ and}$$

$$\text{DSERS}(_{\text{Order.dtd}}) < \text{DSERS}(_{\text{publications.dtd;Order.dtd}})$$

this property is also satisfied by *DSERS(DTD)* metric.

Property 6: $(\exists P)\ (\exists Q)\ (\exists R)\ (|P|=|Q|)\ \&\ (|P; R| \neq |Q; R|)$

This property asserts that we can find two DTD of equal *E(DTD) and DSERS(DTD)* values which when separately concatenated to a same third DTD yields the DTDs of different *E(DTD) and DSERS(DTD)* values. Assume we have three DTD documents; P, Q and R that do not have elements in common i.e. their component definitions for their namespaces [33] are different. The first DTD document P.dtd having 4 equivalence classes consist of similar structured element and the number of elements in each class are C_1=3, C_2=3, C_3=1, C_4=1. The second DTD, Q.dtd has elements completely different in structure from the elements of P.dtd and its equivalence classes with the number of their member elements are C_5=3,C_6=3,C_7=1,C_8=1.The last schema R.dtd has similar elements as P.dtd has, that is it has the same structured elements and hence same graph representation as P.dtd has and its classes are: C_1=3, C_2=3, C_3=1, C_4=1. Note that Q and R do not have any

element in common. We can find that the *E(DTD) and DSERS(DTD)* values for these three documents are equal and found as:

$$E(DTD)_{P,Q,R} = -\sum_{i=1}^{4} P(C_i)\log_2 P(C_i)$$
$$= -[(3/8)*\log_2(3/8) + (3/8)*\log_2(3/8) + (1/8)*\log_2(1/8) + (1/8)*\log_2(1/8)]$$
$$= 1.81128$$

$$DSERS(DTD)_P = DSERS(DTD)_Q = DSERS(DTD)_R = \frac{\sum_{i=1}^{4} de_i^2}{8}$$
$$= (3^2+3^2+1^2+1^2)/8$$
$$= 2.5$$

If the DTD documents P and R are combined then the combined DTD has four classes since these two DTDs have the same graph representation with no common nodes. Hence the number of resulting equivalence classes with their member elements will be C_1=6, C_2=6, C_3=2, C_4=2 and:

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

$$DSERS(DTD)_{(P;R)} = \frac{\sum_{i=1}^{4} de_i^2}{16}$$
$$= (6^2+6^2+2^2+2^2)/16$$
$$= 5$$

If the DTDS, Q and R are combined then the number of resulting classes will be eight since these DTDs also have fully different structured elements. The classes and member elements' counts of combination of Q, R will be C_1=3, C_2=3, C_3=1, C_4=1, C_5=3, C_6=3, C_7=1, C_8=1 and:

$$E(DTD)_{(Q;R)} = -\sum_{i=1}^{8} P(C_i)\log_2 P(C_i)$$
$$= -[(3/16)*\log_2(3/16) + (3/16)*\log_2(3/16) + (1/16)*\log_2(1/16) + (1/16)*\log_2(1/16) + (3/16)*\log_2(3/16) + (3/16)*\log_2(3/16) + (1/16)*\log_2(1/16) + (1/16)*\log_2(1/16)]$$

$$= 2.81128$$

$$DSERS(DTD)_{(Q;R)} = \frac{\sum_{i=1}^{8} de_i^{\,2}}{16}$$
$$= (3^2+3^2+1^2+1^2+3^2+3^2+1^2+1^2)/16$$
$$= 2.5$$

Since

$$E(DTD)_{(P;R)} \neq E(DTD)_{(Q;R)} \text{ and}$$
$$DSERS(DTD)_{(P;R)} \neq DSERS(DTD)_{(Q;R)}$$

this property is satisfied by our *E(DTD)* and *DSERS(DTD)* metrics.

Property 7: There are program bodies P and Q such that Q is formed by permuting the order of the statement of P and (|P| ≠ |Q|).

If we changed the place of the definitions of elements and attributes in the DTD document *E (DTD)* and *DSERS (DTD)* metrics values for the resulting DTD do not change. Although SE and DSERS metrics for XSD satisfy this property *E(DTD)* and *DSERS(DTD)* do not. This due to the fact that in XSD we can define any local element or attribute as global element or attribute and make them reusable components which result *fan-in* and *fan-out* values of these elements to be different in the directed graph representation of XSD. However this is not the case for DTD. In order to confirm the same XML document DTD documents should be designed in the same manner and hence element's design style in DTD cannot be changed. Hence, this property is not satisfied by both *DSERS(DTD)* and *E(DTD)* metrics.

Property 8: The value of *E(DTD) and DSERS(DTD) are* real numbers so renaming of DTD cannot change the values of both metrics. Hence, these two metrics clearly does adhere to this property.

Property 9: $(\exists P)\ (\exists Q)\ (|P| + |Q|) < (|P; Q|)$. By using the same example given under *property 5* this property is not satisfied by *E(DTD)* metric since we find that:

$$E(DTD)_{publications.dtd;Order.dtd} < E(DTD)_{publications.dtd} + E(DTD)_{Order.dtd}$$
$$2.08255 < 1.50588 + 1.35203$$

DSERS(DTD) metric does also not satisfy this property since :

$$DSERS(_{publications.dtd;Order.dtd}) < DSERS(_{publications.dtd}) + DSERS(_{Order.dtd})$$
$$8.91667 < 6.53846 + 6.69231$$

In this section we have validated *E(DTD)* and *DSERS(DTD)* metrics through Weyuker's properties and found that the proposed measures satisfy six Weyuker's property out of nine, which established *E(DTD)* and *DSERS(DTD)* measure as well structured one.

5.7 Empirical Validations of *E(DTD)* and *DSERS(DTD)* Metrics

For empirical validation of these two metric we have analyzed 50 real DTD files from web. The data evaluated after analyzing these files is given in Table 5.2. Note that the data given in Table 5.2 is ordered by Size metric values. The graphs which depict the comparison results of *E(DTD)* metric with Size, Fanning and *TI* metrics are shown in Figure 5.1, 5.2 and 5.3 respectively.

Table 5.2. *Size, Fanning, TI, E(DTD), DSERS(DTD)* metrics values for analyzed DTD files. Last column references web addresses of analyzed DTDs.

ID	Size	Fanning	TI	E(DTD)	DSERS(DTD)	REF
1	4	0.75	0	0.81	2.50	25
2	4	0.75	0	0.75	1.50	92
3	4	0.75	0	1.00	1.00	92
4	6	1.50	0.4	2.58	1.00	25
5	6	0.83	0	2.25	1.33	92
6	7	1.14	0.13	2.81	1.00	92
7	7	0.86	0	2.52	1.29	62
8	7	1.00	0.06	1.66	2.71	25
9	7	1.43	0.27	1.66	2.71	26
10	8	0.88	0	1.75	2.75	28
11	8	1.00	0.02	2.25	1.75	92
12	8	1.13	0.095	2.41	1.75	92
13	8	1.00	0.04	1.55	3.50	62
14	8	1.75	0.33	3.00	1.00	25
15	10	0.90	0	2.72	1.60	92
16	10	0.90	0	1.57	4.20	28
17	10	0.90	0	1.72	3.40	28
18	10	0.90	0	1.57	4.20	28
19	11	0.91	0	2.04	3.18	92
20	11	0.91	0	1.68	3.91	28
21	11	0.91	0	0.78	6.27	28
22	11	1.09	0.04	1.67	4.82	25
23	11	2.09	0.03	2.55	2.27	26
24	11	1.09	0.04	1.62	4.27	27
25	12	1.08	0.04	1.58	5.67	25

Table 5.2 (Cont.)						
26	12	0.92	0	1.21	7.00	28
27	13	1.00	0.02	1.61	4.85	28
28	13	0.92	0	1.57	5.15	25
29	13	0.85	-0.02	2.65	2.69	92
30	13	1.08	0.03	1.82	4.69	27
31	13	0.92	0	2.51	3.23	27
32	13	1.07	0.03	1.51	6.54	25
33	14	0.93	0	1.73	4.71	28
34	14	1.00	0.01	3.09	2.00	26
35	15	0.93	0	2.87	2.60	92
36	15	0.93	0	1.24	8.47	92
37	16	0.88	-0.01	2.35	3.50	28
38	16	1.00	0.09	1.63	7.00	25
39	16	0.94	0	1.19	9.38	93
40	17	0.94	0	1.57	7.00	28
41	17	0.94	0	0.95	11.71	92
42	17	1.12	0.03	1.85	6.65	25
43	17	1.24	0.04	2.13	5.59	25
44	18	0.94	0	0.91	12.67	92
45	18	1.39	0.06	3.17	2.22	92
46	20	2.10	0.13	3.42	1.90	29
47	20	1.35	0.04	4.01	1.60	26
48	32	1.34	0.03	2.18	12.06	27
49	32	1.25	0.02	2.32	9.88	25
50	32	1.19	0.02	2.52	7.63	30

From Table 5.2 and Figure 5.1 it can be observed that *E(DTD)* metric evaluates different values for analyzed DTDs that have the same Size metric values. This is due to the fact; Size metric ignores complexity of DTD caused by frequencies of similar structured elements defined in DTD. The same point is also missed by *TI* and Fanning metrics. Further having negative value for *TI* metric cannot be acceptable according to Weyuker's second property. As can be seen from Figure 5.3 most of DTDs have 0% of *TI* values whatever their size are. In this point of view TI metric fails in differentiating DTDs in terms of their complexities. The last graph shown in Figure 5.4 depicts comparison result between *E(DTD)* and *DSERS(DTD)* metrics. Since these two metrics have inverse relation, lower *E(DTD)* and higher *DSERS(DTD)* values have the same meaning: lower psychological complexity of

DTD. The values of these two metrics are not in contradiction as can be observed from Figure 5.4.

From these observations we can conclude that our proposed DTD metrics are able to differentiate DTDs that have equal number of elements and can be useful to provide feedback about psychological complexity of DTD.

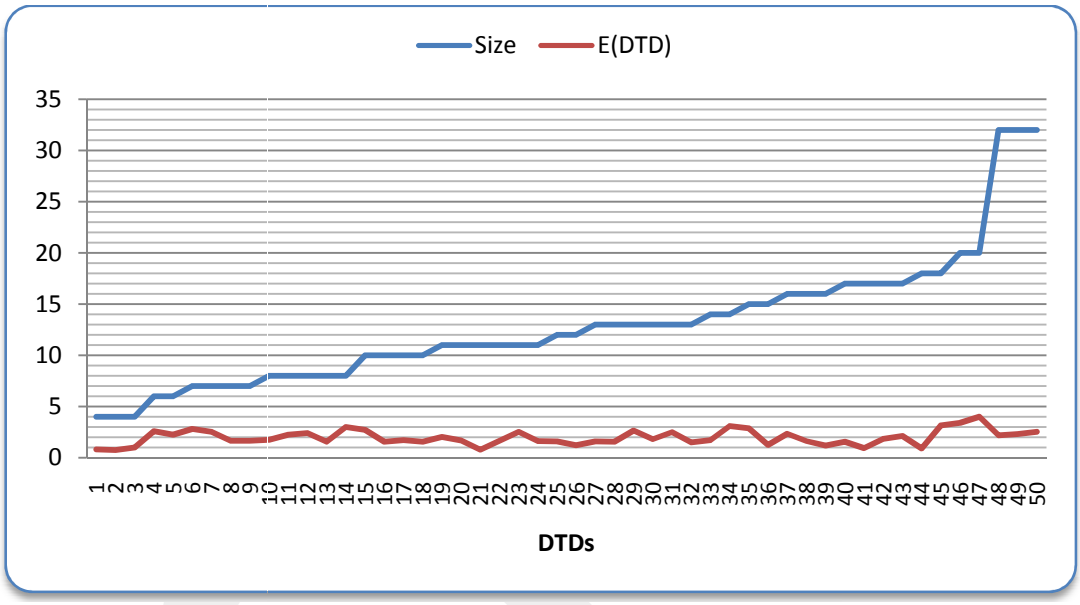

Figure 5.1. E(DTD) vs. Size metrics results. The data is ordered by Size metric values for analyzed DTDs.

.

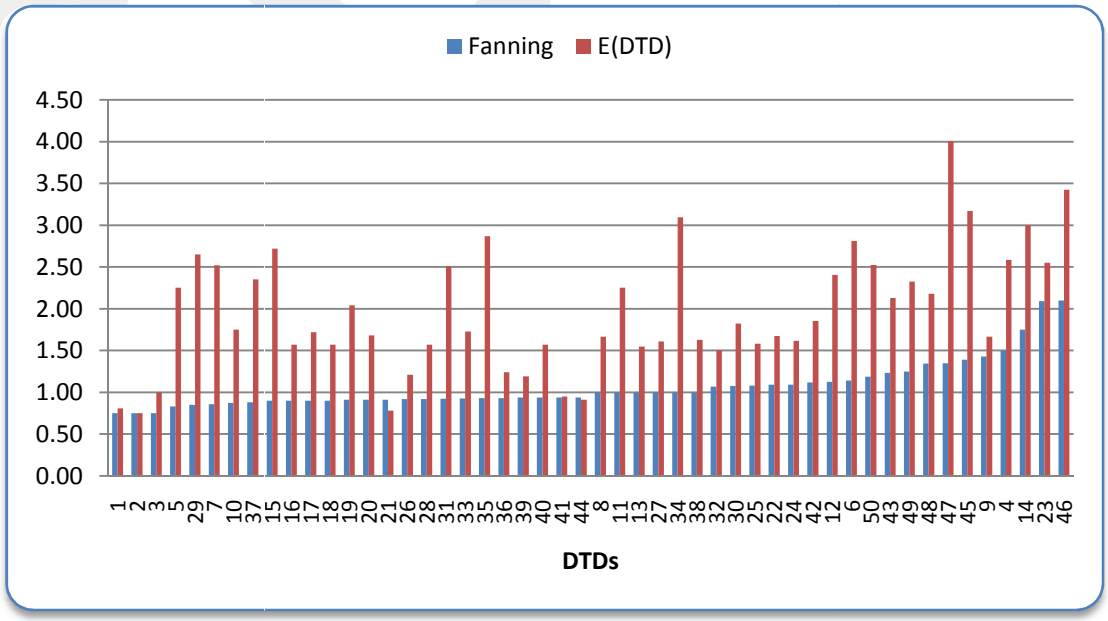

Figure 5.2. E(DTD) vs. Fanning metrics results. The data is ordered by Fanning metric values for analyzed DTDs.

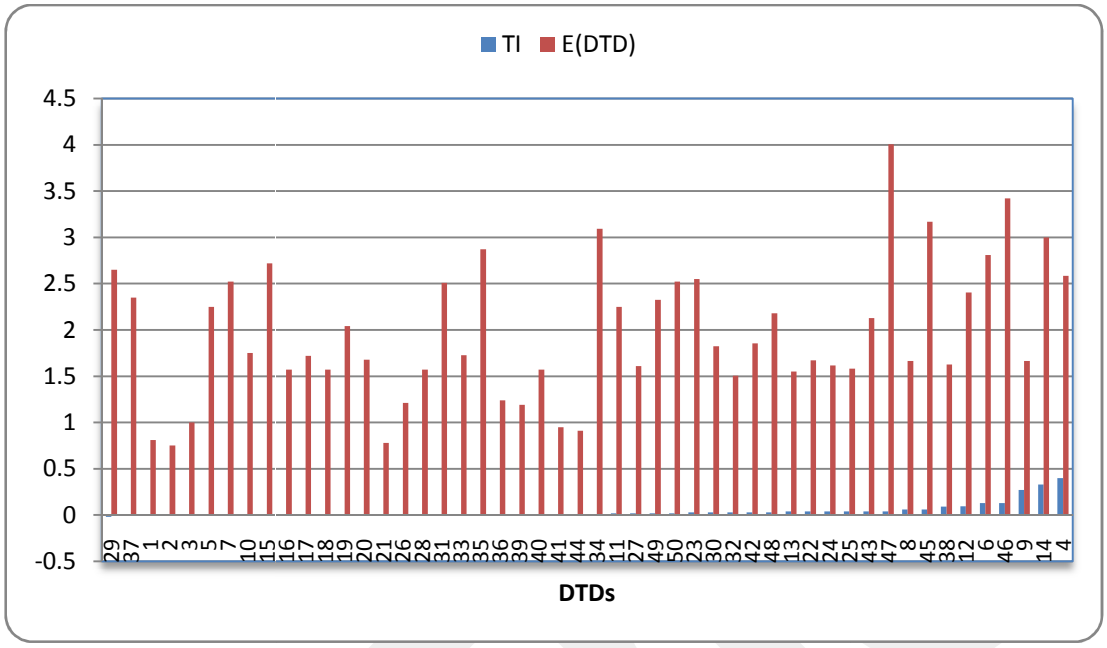

Figure 5.3. E(DTD) vs. TI metrics results. The data is ordered by TI metric values for analyzed DTDs.

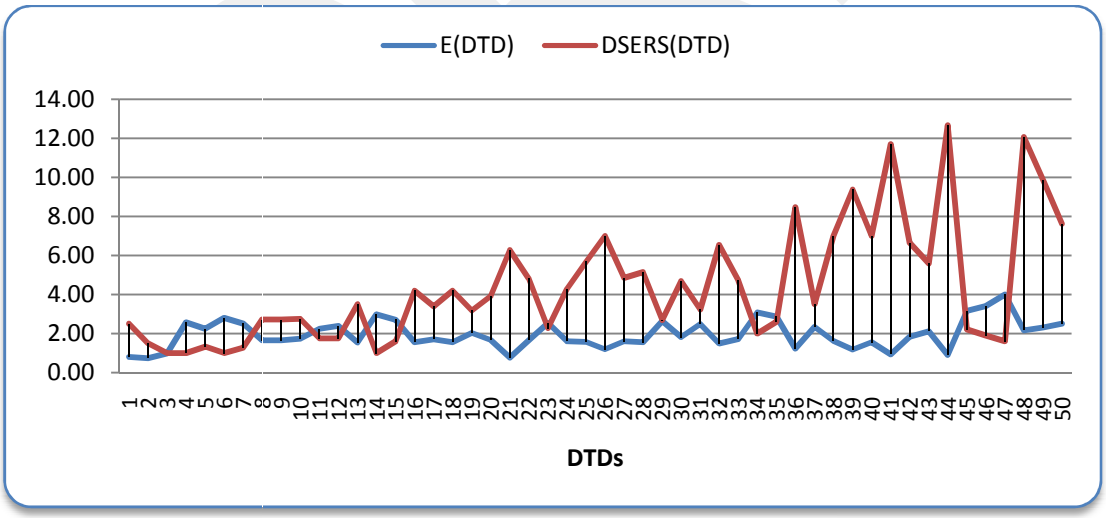

Figure 5.4. DTD vs. DSERS(DTD) metrics results.

5.8 Concluding Remark on *E(DTD)* and *DSERS(DTD)* Metrics

We have presented the complexity metrics for measuring the structural complexity of a given XML schema document written in W3C Document Type Definition language. It was found that measuring the complexity of the XML schema

documents by entropy metric, *E(DTD)*, and *DSERS(DTD)* is more realistic, since the *E(DTD)* and *DSERS(DTD)* metrics calculate the structural complexity of DTDs by considering the fact; the schema documents have less diversity in element are less complex in comparison to the others which have greater diversity. That is, the presented approach has exhibited the better representation of structural complexity of a given schema documents. In order for the proposed metric to be reliably applied for the assessment of the XML schema documents written in DTD it should also be empirically validated. The empirical validations of the proposed metrics have been carried out by collecting 50 DTD documents from the web and comparing the values of the newly proposed metrics with the values of other metrics for these DTDs. From empirical validation results we observed that newly proposed DTD metrics evaluates different complexity values for DTDs that have equal element declarations measured by Size metric. Hence, we can conclude that our metrics can be useful in differentiating DTDs having the same size. .

CHAPTER 6

Related Technology

XML WEB SERVICES

6.1 Introduction

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 [2, 3, 60, and 61] whose implementation details are hidden behind their interfaces. Web services that are based on 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.

As a most widely accepted and successful type of service the XML Web services has become the fundamental building blocks in the movement towards distributed computing on the Internet and providing the platform for integrating diverse applications regardless of where they reside or how they were implemented. By exposing existing applications as XML Web services, the developers are provided a way to build new, more powerful applications that use XML Web services as building blocks.Particularly for the business people Web services have become a powerful means to achieve their business goals. So, for a business person a Web service is a business process or part of a business process that can be made available over a network to internal and/or external business partners to build a new business process or to achieve a business goal. By using open technology (XML and the Internet protocols) and open standards managed by broad consortia such as OASIS[75] and the W3C integrating application functionality within an organization or integrating applications between business partners is achieved more rapidly, easily, and cheaply than ever before. Due to the fact that the XML Web Services are

based on an open standardized suite of technologies such as XML, Hyper Text transport Protocol (HTTP) [59, 77] Simple Object Access Protocol [2, 3, 60, 61, 74] (SOAP), Universal Description, Discovery, and Integration [3, 76] (UDDI) and Web Service Description Language [2, 3, 42, 61] (WSDL) industry-wide support graded the popularity and importance of this platform.

The increasing popularity and acceptance of the XML Web Services led the developers of Web service to make research activities to adopt the best practices of web service implementation and to find the ways for managing web services more effectively.

Particularly, when the need for the integration of various applications to build a new Web process arises, providing the interoperability between these applications becomes main concern of the developers. In this aspect, the degree in flexibility of Web services has an effect on the overall integration process and should be managed effectively. As the complexity in Web service increases the flexibility of it decreases and, in turn, maintaining the Web service will become more of a challenge. Further, the data flow between the participants should be properly managed to provide the consistency in data exchanged between the applications via messaging. Hence in the integration process as the number of applications that are exposed as a Web service increases the management of data flow between services becomes a major issue which can affect the overall quality of the resulting Web process if it is not handled carefully.

It is well known that the maintainability is one of the important factors that affect the quality of any kind of software projects. As a loosely coupled software component the development of Web service also requires modelling, measurement, and quantification for the ease of maintainability purpose. 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.

Although lots of software metrics have been studied for decades in order to improve the quality of the traditional software products, unfortunately, not too much effort has been done 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 a kind of software project. Since the degree in complexity has effect on maintaining any kind of software project we studied in developing metrics to measure the complexity of a Web service.The complexity degree of a Web service can be measured by analyzing 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 characterized 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 analyze the structures of the messages for measuring the data complexity of a Web service via the usage of software metrics. In this chapter we propose a suite of metrics to evaluate and maintain the quality of the Web service in terms of its maintainability.

This chapter is organized as follows. In section 6.2 we give a brief overview for the architecture of Web services and the structure of WSDL documents. Researches related with WSDL metrics are reviewed in section 6.3. The proposed metric is demonstrated by examples in section 6.4. A comparative study with other measures has been done in the same section. The validation of proposed metrics is given in section 6.5. Lastly, section 6.6 provides concluding remarks and a reflection on future work.

6.2 Background

There are several definitions developed for a Web service. The Web services Architecture working group of the W3C [78] 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 serialization 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 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 URL (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. The Figure 6.1[79] illustrates the roles of the service provider, the service requester, and the service registry in the Web service architecture.

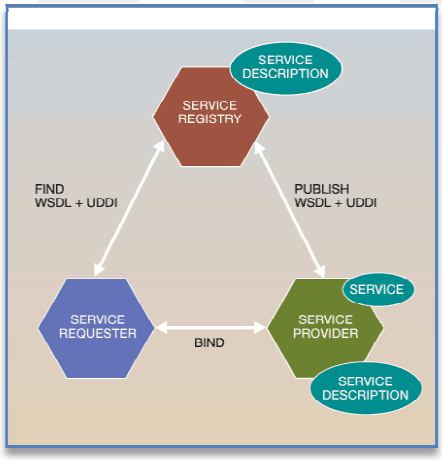

Figure 6.1 The Web services architecture.

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 the problems in 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:

- Simple Object Access Protocol (SOAP)
- Universal Description, Discovery, and Integration (UDDI)

- Web Services Description Language (WSDL)

Figure 6.2[2] illustrates, on a high level, the relationship between these standards.The following subsections provide an overview for each specification.

6.2.1 Simple Object Access Protocol-SOAP

SOAP is the communication protocol intended for exchanging structured information in a decentralized, 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 [74].

6.2.2 Universal Description, Discovery, and Integration -UDDI

The specifications of Universal Description, Discovery and Integration (UDDI) [76] sponsored by OASIS [75] define a registry service for Web services and for other electronic and non-electronic services [76]. 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.

6.2.3 Web Service Description Language-WSDL

All of the information necessary to invoke a Web service should be clearly described and be made available in a way that the service consumers can easily access and process the information about the service. For this purpose Web Services Description Language provides a model and an XML format for describing Web services. WSDL 1.1[42] was submitted to the W3C in late 2001, and the W3C Web Service Description Working Group was formed in early 2002 to standardize WSDL. WSDL 2.0 is the standard version of WSDL that includes significant changes and improvements [70]

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 such as “how” and “where” that functionality is offered.

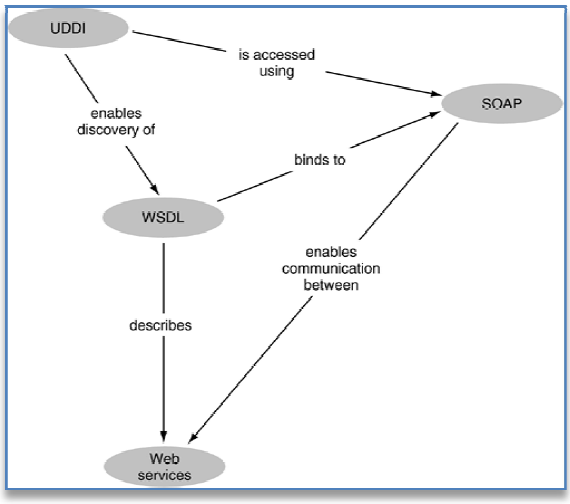

Figure 6.2 The relationships between web service standards.

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 diverse set of programming languages in order to work in 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, in order for accommodation of these changes building a flexible document structure that can be extended will pay off in the future. In [63] the use of WSDL and XML Schema is mandated for describing Web services. By this way the interoperability at the service description layer is ensured.

6.2.4 XML Schema

As we mentioned earlier, data flow between applications should be properly maintained to provide consistency in application's data conveyed by the input and output messages of the Web services in interaction. In order to provide a common data type systems and to represent application's data the WSDL document is supported by XML Schema which is either imported to or embedded into the WSDL document and encapsulated by <types> construct of WSDL. The decision in XML Schema design to represent the application's data exchanged may affect the understandability of the exchanged messages described in WSDL document and in turn comprehending of a Web service defined by WSDL document.

6.3 Related Work

In terms of measuring the complexity of a Web service Yu et al [68] 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 [68] are reviewed below.

Argument per Operation (APO): Given 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 defined as:

$$\mathrm{APO} = \frac{n_a}{n_o} \qquad 18.$$

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:

$$\mathrm{DAR} = \frac{\mathrm{DAC}}{n_a} \qquad 19.$$

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

$$\mathrm{ARS} = \frac{\sum_{a \epsilon A} |a|^2}{|A|} \qquad 20.$$

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 *WMC* (*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 which shows the inheritance relation of the type definitions for the arguments.

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

6.3.1 Discussion

According to Yu *et al* [68] while lower *ARS* indicates more specialized functionality of a Web service higher *DAR* indicates more consistency of argument declarations.

```
...
<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>
...
```

Listing 6.1 Example WSDL document.

```
.....
<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>
.....
```

Listing 6.2 Example WSDL document.

Consistent argument declarations make it easier to understand and reuse the components. Further, they claim that having higher *DAR* and lower *ARS* therefore, WSDL interfaces more reusable in general, but also more cumbersome to understand the functionality of each operation. 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 [43], 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 listing 6.1 and listing 6.2, both offer only one operation having 3 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 listing 6.1, 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 [68]. The second WSDL in listing 6.2, on the other hand, has also *DIT* value of 1 and *NOC* value of 1.

Based on these two WSDL documents both services have equal *APO, DAR, ARS, OPS, DIT, NOC* values which are not sufficient to differentiate these two services in terms of their complexities.

The APO metric cannot capture the difference between these two services since it ignores the complexity of the arguments' data structure. For the same reason, *DAR* and ARS fail to differentiate these two services since both *DAR* and *ARS* metrics do not consider the complexities of data types due to their internal architectures. While 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 due to the variety in the contents of the complex types defined in XSD.

6.4 Proposed Metrics

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. Because the execution of an operation requires to exchange 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 is also under consideration. In this point of view understandability of message structures has an important role in comprehending the operations that a Web service offers.

The arguments that an operation takes are contained by 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 etc. definition/declarations in order 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 will also affect the understandability of the message structures. As defined before the data complexity of a Web service refers 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 analyzing the XSD embedded in WSDL and the structures of its requesting and responding messages used by the operations a Web service offers.

In section 4.2 we presented a complexity metric *C(XSD)* [67] for the assessment of the quality of XML Schema. The complexity of XSD document measured by the presented metric is evaluated by summing up all of its components' complexities. In this evaluation each component's complexity degree is reflected by a weight value which is assigned based on component's internal architecture.

The complexity degree of each XSD component will affect 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 [68] and becomes our main focus to measure data complexity of a Web service.

By analyzing the structures of the exchanged messages described in WSDL we measured the data complexity of a given Web service via the newly proposed metrics introduced in the following subsections. We have also done an empirical validation of our proposed metrics through analyzing 56 different real WSDL documents written in WSDL 1.1[42]. Additionally, to verify the soundness and robustness of our metrics a comparative study with similar measures [68] has been done as a part of empirical validation process and the statistics (see Table 1 in Appendix) referring these measurement results has been collected.

The analyzed WSDL documents are collected by searching some well known sites *http://www.xmethods.com/, http://www.webservicelist.com/* that list publicly available web services. The references to these analyzed WSDL documents are given in Table 2 in Appendix C.

6.4.1 Data Weight of a WSDL (DW(wsdl))

The Data Weight of a given WSDL can be defined as the sum of the data complexities of each input and output messages. By analyzing the message structures which contain the arguments that the operations of a Web service take we can measure each message's data complexity.

Intuitively, one may expect that as the number of operations and the number of arguments increase the understandability of a Web service becomes more difficult. The *APO* (*Arguments per Operation*) and *OPS* metrics presented in [68] were intended to measure the interface complexity of a given Web service. However, we claim that this is not always the case since 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. The effort required to understand the XSD components has been discussed in section 4.2 when we presented *C(XSD)* metric.

In order to measure the data complexity of a given message we evaluate the data complexities of its part elements that describe the arguments. The data complexity of each part element can be reflected by assigning a weight value which is based on the

complexity due to the data structures of arguments. As we presented in section 4.2, based on the internal architectures the complexity degree of the elements, types that are defined/declared in the XSD may vary and are assigned a weight value reflecting their complexity levels. Since these XSD components are associated with the arguments to describe the arguments' data structures, it is meaningful to assign a weight value to these arguments which are equal to their associated XSD components' weight values so that the arguments' complexities due to their data structures are reflected.

By summing the weight values of each part elements belonging to the message construct of WSDL the data complexity of a message can be evaluated. 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 Data Weight of WSDL *(DW (wsdl)*) metric to capture the complexity of a Web service due to the data flowed 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, Data Weight of WSDL metric is defined as

$$DW(wsdl) = \sum_{i=1}^{n_m} C(M_i) \quad 21.$$

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

$$C(M) = \sum_{j=0}^{n_p-1} wp_i \quad 22.$$

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 we 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 which reflects their complexity degrees and is associated to the arguments, the wp_i has the same weight value with the associated element or type definitions of XSD. That is

wp = we, *if part has element reference to the Schema element declaration* 23.

= wt, *if part has type reference to the Schema element definition* 24.

= 0, *if the message has no <part> element.* 25.

where *we* is the weight of the associated element declaration and defined by the equations 12.1-12.6 in section 4.2.2, *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 ,*if the type is complex type definition* 26.

= ws, if the type is simple type definition 27.

where *wc* is given by equation 16.1 and *ws* is defined by equations 17.1 through 17.4 in section 4.2.2.

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

Example 6.1: In listing 6.3 and listing 6.4 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 a XML Schema type *string*. In order to evaluate *DW(wsdl)* we first calculate the complexities of its each message.

$$\mathrm{C}(\text{“}GetGSInformationRequest\text{”}) = \sum_{j=0}^{1} wp_j$$

$$= \mathrm{wp}_{\mathrm{ID}} + \mathrm{wp}_{\mathrm{Password}}$$

$$= \mathrm{wt}_{\mathrm{ID}} + \mathrm{wt}_{\mathrm{Password}}$$

$$= \mathrm{ws}_{\mathrm{string}} + \mathrm{ws}_{\mathrm{string}}$$

$$= 1 + 1$$

$$= 2$$

$$\mathrm{C}(\text{“}StringResponse\text{”}) = \sum_{j=0}^{0} wp_j$$

$$= \mathrm{wp}_{\mathrm{Response}}$$

$$= \mathrm{wt}_{\mathrm{Response}}$$

$$= \mathrm{ws}_{\mathrm{string}}$$

$$= 1$$

```
....
<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>
.....
```

Listing 6.3 Example WSDL document.

The weight value for a part element is assigned based on the weight value of the argument it describes and the argument described by part element will have a weight of its associated type of XML Schema. By equation 17.1 in section 4.2.2 the weight values for built-in simple types of XML Schema are assigned to 1. Consequently, by equation 27, 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 listing 6.3 is evaluated by:

$$\begin{aligned} \mathrm{DW(wsdl)} &= \sum_{i=1}^{2} C(M_i) \\ &= \mathrm{C}(\text{“}\textit{GetGSInformationRequest}\text{”}) + \mathrm{C}(\text{“}\textit{StringResponse}\text{”}) \\ &= 2 + 1 \\ &= 3 \end{aligned}$$

In listing 6.4 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 typed GetLandmarkTypes element of the XSD, the output message's part element associates the complex typed GetLandmarkTypesResponse element of XSD for defining the output arguments namely ShapeType and Type. The complexity values for each message evaluated as:

$$C(\text{“}GetLandmarkTypesSoapIn\text{”}) = \sum_{j=0}^{0} wp_j$$
$$= wp_{parameters}$$
$$= we_{tns:GetLandmarkTypes}$$
$$= wc_{complexType}$$
$$= 1$$

$$C(\text{“}GetLandmarkTypesSoapOut\text{”}) = \sum_{j=0}^{0} wp_j$$
$$= wp_{parameters}$$
$$= we_{tns:GetLandmarkTypesResponse}$$
$$= wc_{tns:GetLandmarkTypesResponse}$$
$$= 6$$

The method for calculating a complexity value for an element declaration in Schema is presented in section 4.2.2. Based on the equations 12.1-12.6 in section 4.2.2 the data complexity i.e. weight value for the element declaration tns:GetLandmarkTypes of XSD given in listing 6.4 is found as 1. Similarly, the data complexity value for the element declaration tns:GetLandmarkTypesResponse in XSD which is associated for the output arguments description of the operation is found as 6. Hence the overall data complexity value for WSDL document given in listing 6.4 is:

$$DW(wsdl) = \sum_{i=1}^{2} C(M_i)$$
$$= C(\text{“}GetLandmarkTypesSoapIn\text{”}) + C(\text{“}GetLandmarkTypesSoapOut\text{”})$$
$$= 1 + 6$$
$$= 7$$

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 6.1, even both WSDLs have equal *APO, OPS, DA*R and *ARS* values, *DW(wsdl*) values of them are not equal due 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 of *DW* we collected statistics (see Table 1 in Appendix B) by analyzing 56 different WSDL files. We also compared *DW* with *APO* and *OPS* metrics to verify its usefulness as part of empirical validation process of DW metric. The graph shown in Figure 6.3 depicts

the comparison result between the *OPS* and *DW(wsdl)* metricsand Figure 6.4 shows *DW(wsdl)* and *APO* comparison result for analyzed 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.

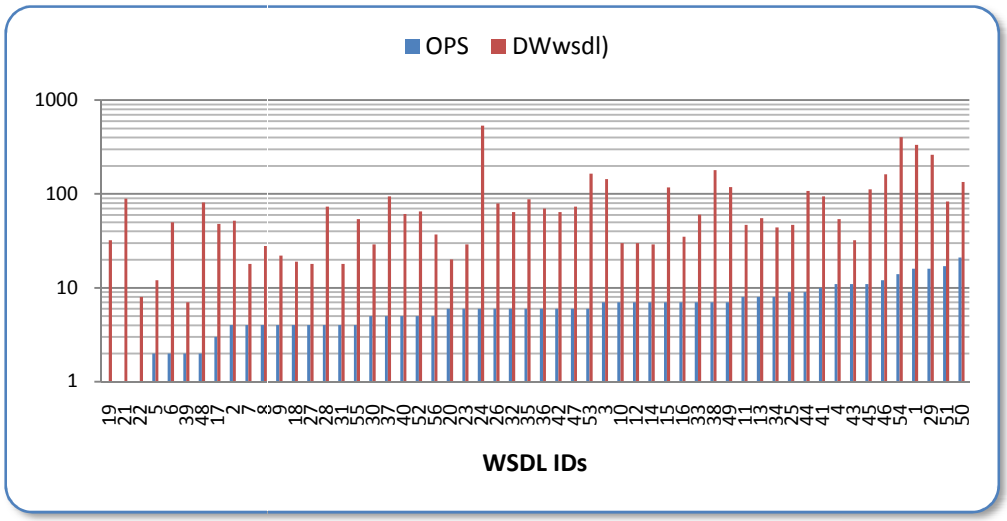

Figure 6.3 The graph drawn according to values of *DW* and OPS metrics for analyzed WSDLs.

It can be observed from these two figures as the number of arguments or operations increases the data complexity of a Web service does not always increases accordingly. This result verifies our claim which we stated at the beginning of this section.

```
<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>
.....
```

Listing 6.4 Example WSDL document.

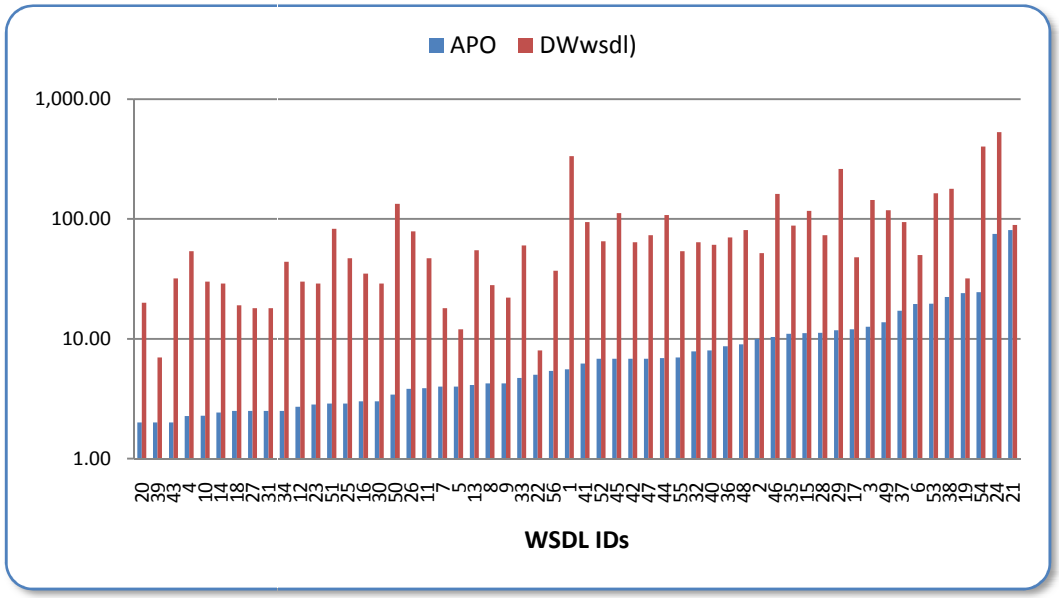

Figure 6.4 The graph drawn according to values of *DW* and APO metrics for analyzed WSDLs.

6.4.2 Distinct Message Ratio (DMR) Metric

Consider the real life WSDL document shown in listing 1 in Appendix A. 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. We can regard these messages have the same structures since the number of arguments they contain and the data complexity values due 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 due to 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 Distinct Message Ratio(*DMR*) metric which is derived from Distinct Message Count (*DMC*) metric to capture this aspect. We first provide a definition for *DMC* metric which considers structures of the messages and adapted from *DAC* metric. Note that while *DAC* metric considers on distinct arguments but ignores the complexities of them, *DMC* considers distinct messages structures characterized by according to the

number of arguments they contain and the data complexities of the contained arguments.

Definition 2: (*DMC*) The distinct message count 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 WSDL document in listing 1 in Appendix A are represented by labelled pairs in order 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 which is equal to the number of pairs and the pairs are labelled as G1, G2 ...G7.

Definition 3: Distinct Message Ratio *(DMR)* can reflect the data complexities of a Web service due to having similar structured messages exchanged by the operations and is defined as:

$$\mathrm{DMR} = \frac{\mathrm{DMC}}{\mathrm{n_m}} \qquad 28.$$

where $\boldsymbol{n_m}$is the total number of messages in WSDL.

Example 6.2: The *DMR* metric value of WSDL document shown in listing 1 in Appendix A is found as:

$$\mathrm{DMR} = 7/21$$
$$= 0.33$$

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. As part of the empirical validation of *DMR* we analyzed 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 Figure 6.5 the graph drawn according to the collected statistics (see Table 1 in Appendix B) depicts the comparison result between *DMR* and *DW*. The graphs in

Figure 6.6 and Figure 6.7 show comparison result between *DMR* and *APO*, and between *DMR* and *OPS* metrics respectively.

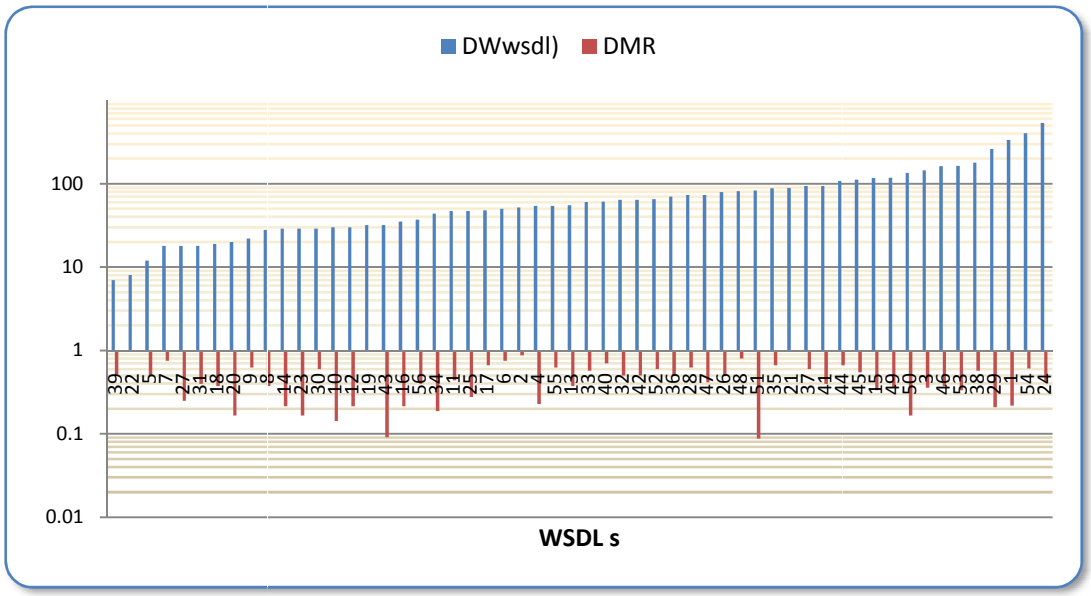

Figure 6.5 The graph drawn according to values of *DW* and *DMR* metrics for analyzed WSDLs.

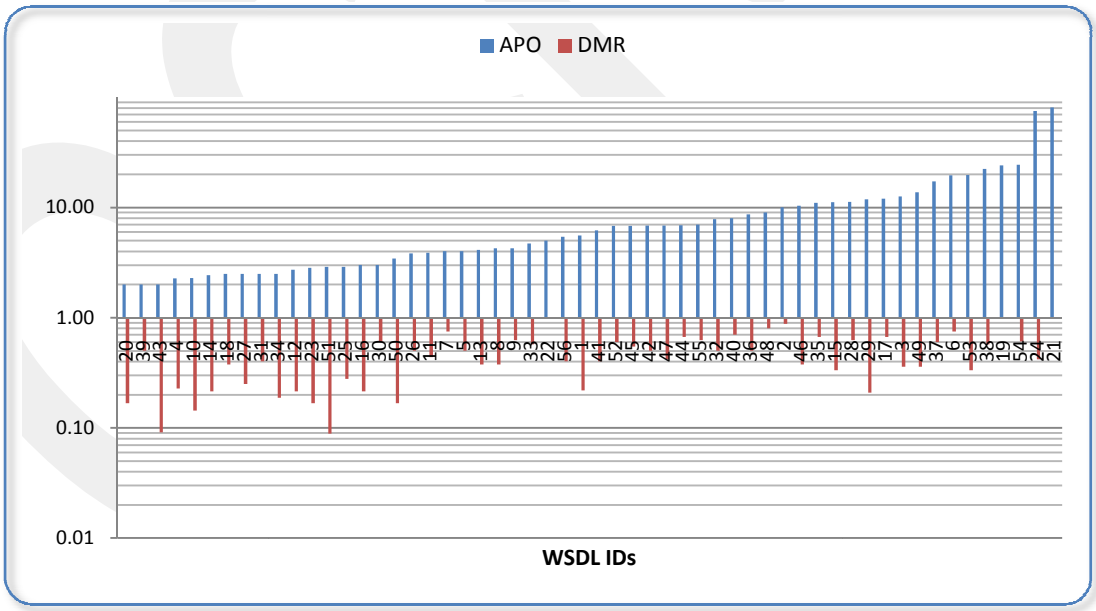

Figure 6.6 The graph drawn according to values of *DMR* and *APO* metrics for analyzed WSDLs.

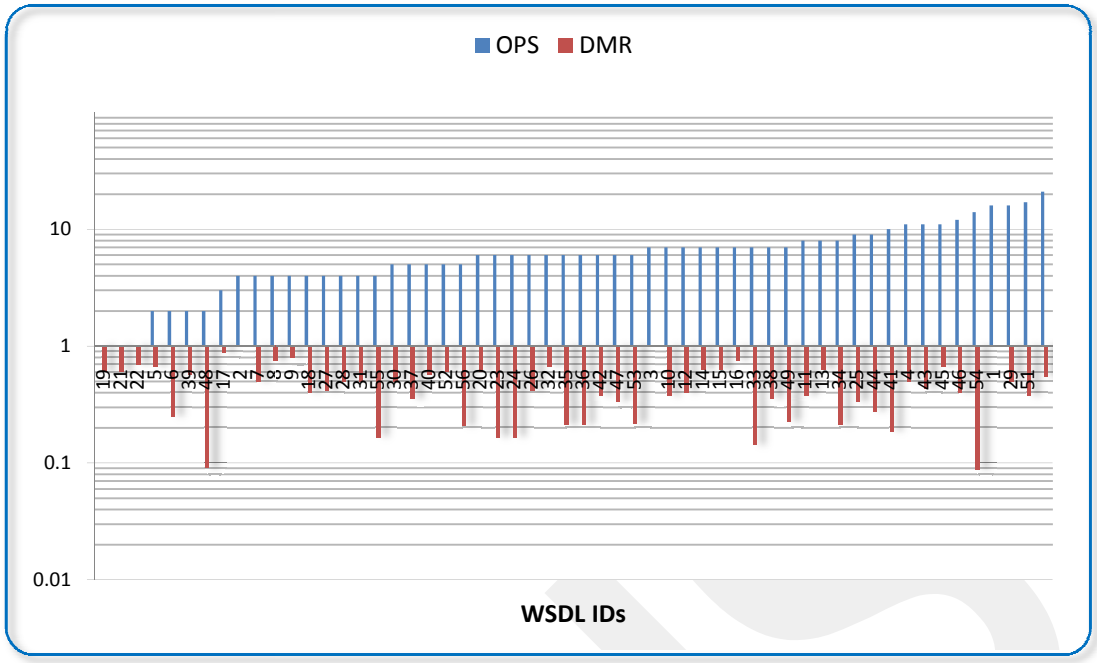

Figure 6.7 The graph drawn according to values of *DMR* and OPS metrics for analyzed WSDLs.

6.4.3 Message Entropy (ME) Metric

Now, let us assume we have two WSDL documents each having twelve 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 messages. It is clear that comprehending the first WSDL is easier than that of the second one due to the fact that repetition of the same structured messages makes the developer more familiar to 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 due to the repetition of similar structured messages.

The Message Entropy (*ME*) metric is intended to measure the complexity due to occurrences of similar structured messages. The concept of entropy has been discussed in section 4.3 when we presented Schema Entropy metric for XSDs,(*SE*). By the similar approach we adapted entropy 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*) Message Entropy of a given WSDL is

$$\mathrm{ME(wsdl)} = -\sum_{i=1}^{DMC} P(dm_i) log_2 P(dm_i) \quad 29.$$

$$\mathrm{P}(dm_i) = \frac{nom_i}{n_m} \quad 30.$$

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

Example 6.3: The WSDL file shown in listing 1 in Appendix A has seven distinct messages structures represented by the set of pairs G1, G2, … G7, i.e. 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 which gives the occurrences of the pair G3. The classes that consist of the same pairs and number of their members listed below:

$$\mathrm{G1} = 3(\mathrm{C(M)}{=}1,\mathrm{arg}{=}1)$$
$$\mathrm{G2} = 4(\mathrm{C(M)}{=}14,\mathrm{arg}{=}9)$$
$$\mathrm{G3} = 8(\mathrm{C(M)}{=}4,\mathrm{arg}{=}3)$$
$$\mathrm{G4} = 2(\mathrm{C(M)}{=}2,\mathrm{arg}{=}1)$$
$$\mathrm{G5} = 1(\mathrm{C(M)}{=}13,\mathrm{arg}{=}9)$$
$$\mathrm{G6} = 2(\mathrm{C(M)}{=}3,\mathrm{arg}{=}1)$$
$$\mathrm{G7} = 1(\mathrm{C(M)}{=}3,\mathrm{arg}{=}2)$$

and by 29 *ME* value for the WSDL file is:

$$\mathrm{ME}(\text{“}u7.wsdl\text{”}) = -\sum_{i=1}^{7} P(dm_i) log_2 P(dm_i)$$

$$= -[(3/21)*\log_2(3/21) + (4/21)*\log_2(4/21) + (8/21)*\log_2(8/21) + (2/21)*\log_2(2/21) + (1/21)*\log_2(1/21) + (2/21)*\log_2(2/21) + (1/21)*\log_2(1/21)]$$
$$= 2.45161$$

In Figure 6.8 the graph depicts the comparison result between *DMR* and *ME* metrics which is evaluated according to the collected statistics (see Table 1 in Appendix B) of analyzed 56 WSDL documents. From the graph 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. We also compared the *ME* metric with *APO* and *OPS* metrics to verify its usefulness, the graphs in Figure 6.9 and Figure 6.10 show the results. From the graphs depicted in Figure 6.9 and Figure 6.10 we can see that complexities of WSDLs which have equal APO and OPS metrics values are different evaluated by ME metric.

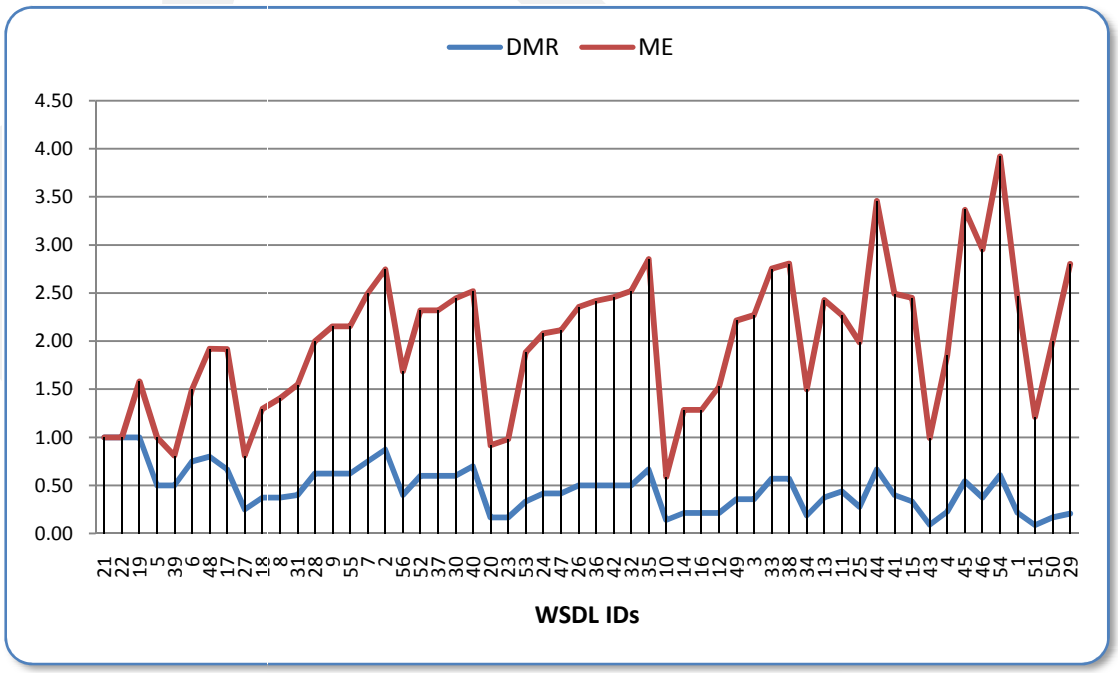

Figure 6.8 The graph drawn according to values of *ME* and *DMR* metrics for analyzed WSDLs.

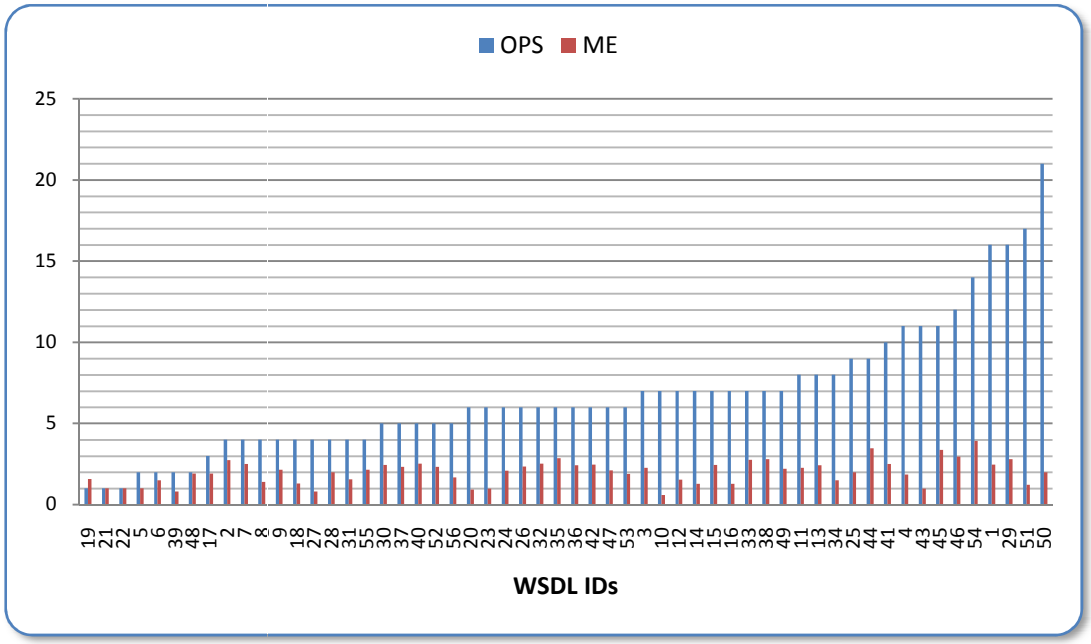

Figure 6.9 The graph drawn according to values of *ME* and *OPS* metrics for analyzed WSDLs.

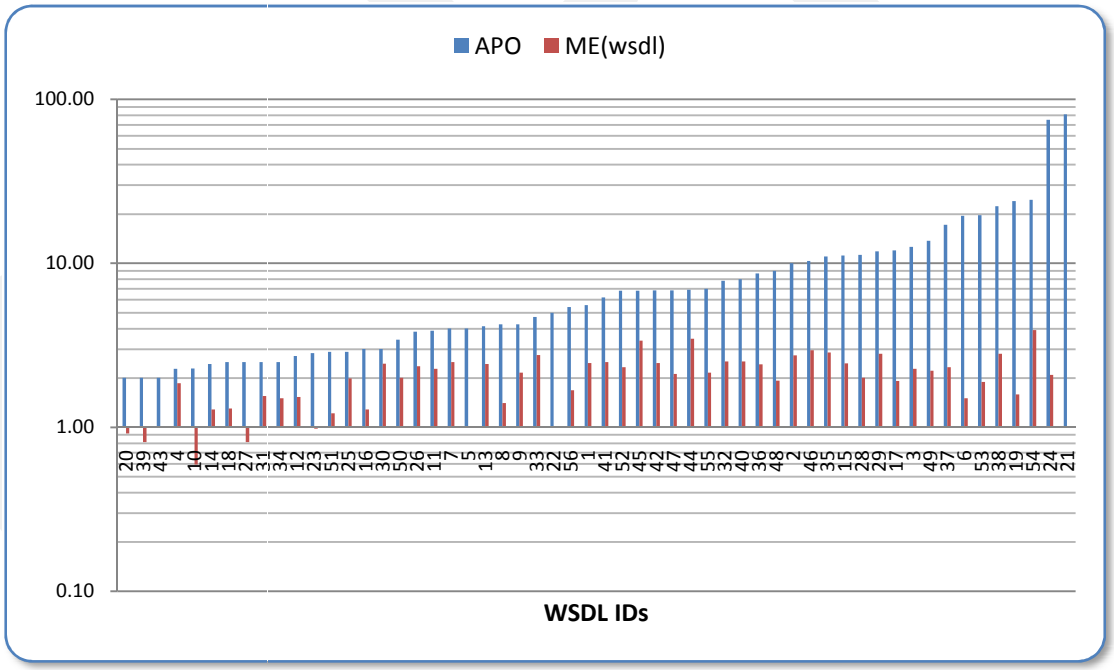

Figure 6.10 The graph drawn according to values of *ME* and *APO* metrics for analyzed WSDLs.

6.4.4 Message Repetition Scale (MRS) Metric

As *ME(wsdl)* metric Message Repetition Scale *(MRS*) can be used for measuring the data complexity of WSDL documents and can be defined as:

$$MRS = \frac{\sum_{i=1}^{n_m} nom_i^2}{n_m} \qquad 31.$$

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

Example 6.4: For the WSDL file shown in listing 1 in Appendix A the *MRS* value is calculated by considering the messages listed in *example 6.3*:

$$MRS = \frac{\sum_{i=1}^{n_m} nom_i^2}{n_m}$$
$$= (3^2+4^2+8^2+2^2+1^2+2^2+1^2)/21$$
$$= 83/21$$
$$= 3.95238$$

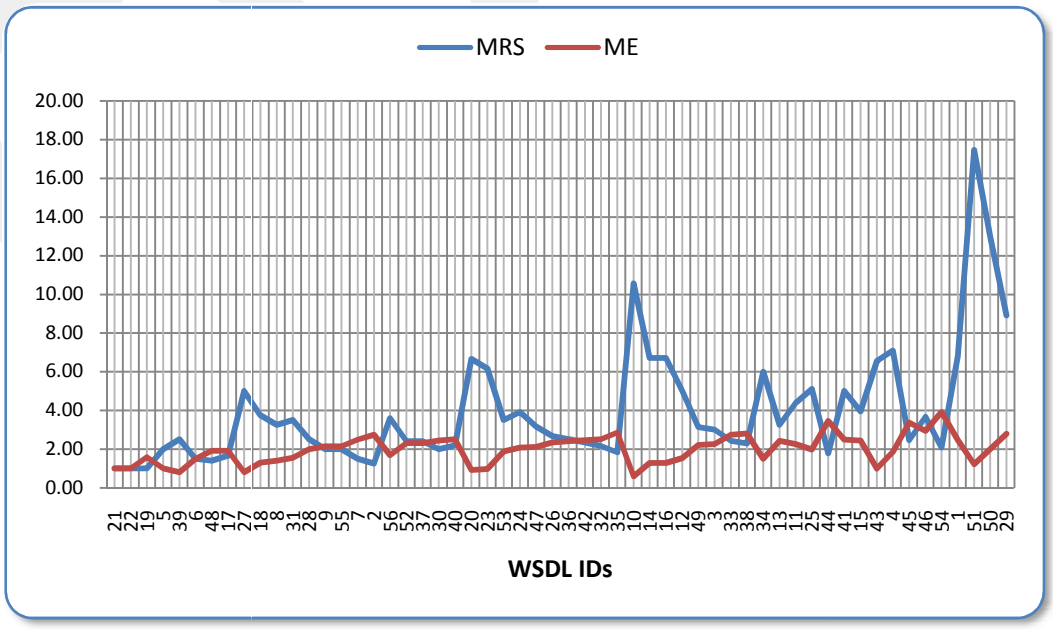

Figure 6.11 The graph drawn according to values of *MRS* and *ME* metrics for analyzed WSDLs.

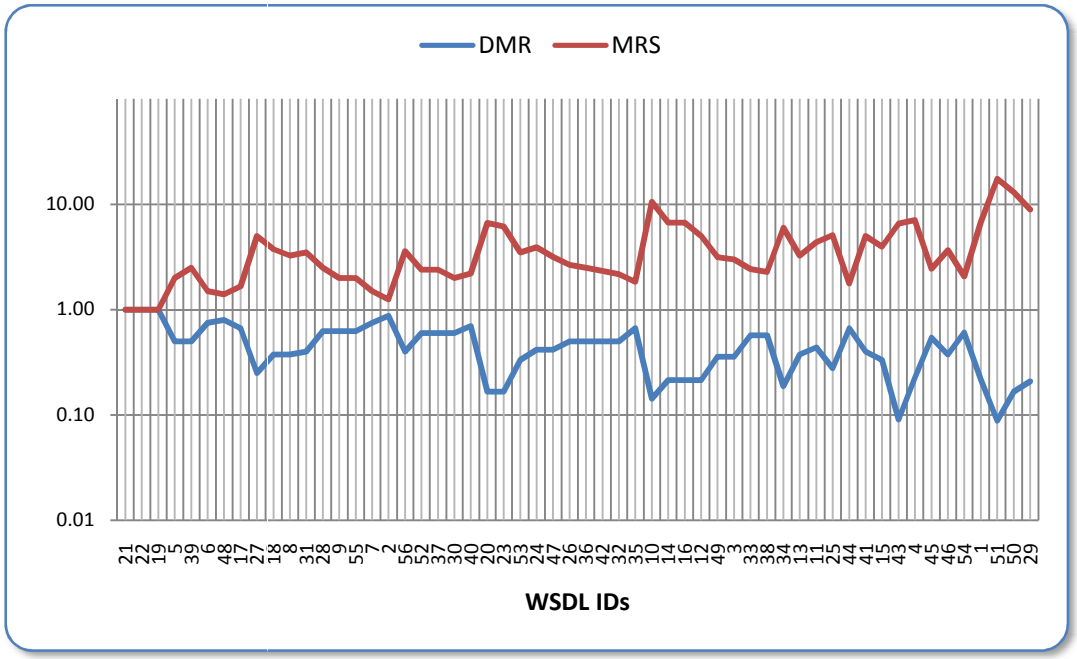

Figure 6.12 The graph drawn according to values of *MRS* and *DMR* metrics for analyzed WSDLs.

We evaluate the graph depicted in Figure 6.11 to show the relation between *MRS* and *ME* after analyzing 56 WSDL documents. As seen from the graph there is inverse relation between *MRS* and *ME* metrics. Additionally, *MRS* is compared with *DMR* and the graph in Figure 6.12 shows that these two metrics are also inversely proportional to each other.

6.5 Theoretical Validations of Proposed Metrics

In this section we evaluate our proposed metrics with the help of Weyuker's properties. In evaluation process we use three example WSDL documents given in listing 6.5, listing 6.6 and listing 6.7 respectively.

The WSDL document given in listing 6.7 is designed by concatenating the WSDLs given in listing 6.5 and listing 6.6. 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. Note that the method for representation of the messages by (*C (M), arg*) pair is explained in section 6.4.2. Having labelled these messages DMC values for these three WSDL are found as 2, 2, and 3 respectively.

```
<?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>
```

Listing 6.5 Example WSDL document.

```
        <?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>
```

Listing 6.6 Example WSDL document.

```
<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>…
```

Listing 6.7 Combination of WSDLs given in listing 6.5 and listing 6.6.

6.5.1 Evaluation of Proposed Metrics by Weyuker's Properties

Property 1: (∃P) (∃Q) (| P | ≠ | Q |). Where P and Q are programme body. As can be observed from Table 1 in Appendix B it is obvious that for different Schema documents different *DW*, *DMR*, *ME* and *MRS* values are evaluated, 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 upon the structures of messages and their 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|.
As can be seen from Table 1 in Appendix B, different WSDL documents may have same *DW*, *DMR*, *ME* and *MRS* values hence, this property is also hold.
Property 4: (∃P) (∃Q) (P≡Q & | P | ≠ | 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 due 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.

Property 5: (∀P) (∀Q) (|P| ≤ |P; Q| and |Q| ≤ |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 listing 6.5, listing 6.6 and listing 6.7 representing P, Q and P;Q. Note that the WSDL document given in listing 6.7 is the combination of the WSDL given in listing 6.5 and listing 6.6. The metrics values given in Table 6.1 for these three WSDL documents are calculated as

$$DW(\text{“}\textit{listing 6.5.wsdl}\text{”}) = \sum_{i=1}^{2} C(M_i)$$

$$= C(\text{“}\textit{GetAddressRequest”}) + C(\text{“}\textit{GetAddressResponse”})$$

$$= wp_{name} + wp_{address}$$

$$= ws_{string} + wc_{Adress}$$

$$= 1 + 5$$

$$= 6$$

$$DW(\text{“}\textit{listing 6.6.wsdl}\text{”}) = \sum_{i=1}^{2} C(M_i)$$

$$= C(\text{“}\textit{GetPhoneRequest”}) + C(\text{“}\textit{GetPhoneResponsee”})$$

$$= wp_{name} + wp_{phone}$$

$$= ws_{string} + wc_{Phone}$$

$$= 1 + 4$$

$$= 5$$

$$DW(\text{“}\textit{listing 6.7.wsdl}\text{”}) = \sum_{i=1}^{4} C(M_i)$$

$$= C(\text{“}\textit{GetAddressRequest”}) + C(\text{“}\textit{GetAddressResponse”})$$

$$+ C(\text{“}\textit{GetPhoneRequest”}) + C(\text{“}\textit{GetPhoneResponsee”})$$

$$= wp_{name} + wp_{address} + wp_{name} + wp_{phone}$$

$$= ws_{string} + wc_{Adress} + ws_{string} + wc_{Phone}$$

$$= 1+5+1 + 4$$

$$= 11$$

$$DMR_{\textit{listing 6.5.wsdl}} = DMR_{\textit{listing 6.6.wsdl}}$$

$$= \frac{DMC}{n_m}$$

$$= 2/2$$

$$= 1$$

$$DMR_{\textit{listing 6.7.wsdl}} = 3/4 = 0.75$$

$$ME(\text{“}\textit{listing 6.5.wsdl}\text{”}) = ME(\text{“}\textit{listing 6.6.wsdl}\text{”})$$

$$= -[(1/2)*\log2(1/2)+(1/2)*\log2(1/2)]$$

$$= 1$$

$$ME(\text{“}\textit{listing 6.7.wsdl}\text{”}) = -[(1/4)*\log2(1/4)+(2/4)*\log2(2/4) + (1/4)*\log2(1/4)]$$

$$= 1.5$$

MRS(*"listing 6.5.wsdl"*) = MRS(*"listing 6.6.wsdl"*)

$= (1^2+1^2)/2 = 1$

MRS(*"listing 6.7.wsdl"*) $= (1^2+2^2+1^2)/4 = 1.5$

As can be observed from Table 6.1 except for DMR metric all other three metrics satisfy this property.

Table 6.1 The metrics values of WSDL documents.

WSDL	*#ofMessages*	*DMC*	*DW*	*DMR*	*ME*	*MRS*
Listing 6.5	*2*	*2*	*6*	*1*	*1*	*1*
Listing 6.6	*2*	*2*	*5*	*1*	*1*	*1*
Listing 6.7	*4*	*3*	*11*	*0.75*	*1.5*	*1.5*

Property 6: $(\exists P)\ (\exists Q)\ (\exists R)\ (|P|=|Q|)\ \&\ (|P; R| \neq |Q; R|)$

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 4 distnict message strutures and its messages are labelled as $G1$, G_2, G_3, G_4. The number of occurrences of these messages is 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 due to the complexity of its part element's data structure defined in the Schema (see definition 2 in section 6.4.2). The second WSDL, Q.wsdl, has messages completely different in structure from the messages of P.wsdl and its messages' with the number of occurences are $G_5=3$, $G_6=3$, $G_7=1$, $G_8=1$.The last WSDL, R.wsdl has similar message structures as P.wsdl has and its messages are labelled as: $G_1=3$, $G_2=3$, $G_3=1$, $G_4=1$. 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 which is combination of P; R and Q; R will be:

$$DW_{P;R}=DW_{Q;R}$$

Hence, DW metric does not satisfy this property.

Since P and R have similar message structures the combined WSDL has 4 different message structures (G_1=6, G_2=6, G_3=2, G_4=2) in total, so;

$$ME_{P;R}=-\sum_{i=1}^{4} P(dm_i)log_2P(dm_i)$$
$$=-[(6/16)*\log_2(6/16)+(6/16)*\log_2(6/16)$$
$$+(2/16)*\log_2(2/16)+(2/16)*\log_2(2/16)]$$
$$=1.81128$$

The WSDLs Q and R have different message structures and combined WSDL has 8 message structures namely G_1=3, G_2=3, G_3=1, G_4=1, G_5=3, G_6=3, G_7=1, G_8=1. So,

$$ME_{Q;R}=-\sum_{i=1}^{8} P(dm_i)log_2P(dm_i)$$
$$=-[(3/16)*\log_2(3/16)+(3/16)*\log_2(3/16)$$
$$+(1/16)*\log_2(1/16)+(1/16)*\log_2(1/16)$$
$$+(3/16)*\log_2(3/16)+(3/16)*\log_2(3/16)$$
$$+(1/16)*\log_2(1/16)+(1/16)*\log_2(1/16)]$$
$$=2.81128$$

$ME_{P;R}\neq ME_{Q;R}$ hence *ME* metric holds this property.

$$DMR_{P;R}=4/16=0.25$$
$$DMR_{Q;R}=8/16=0.5$$

$DMR_{P;R}\neq DMR_{Q;R}$, hence *DMR* metric does adhere to this property.

$$MRS_{P;R}=\frac{\sum_{i=1}^{4} nom_i^2}{16}$$
$$=(6^2+6^2+2^2+2^2)/16$$
$$=80/16$$
$$=5$$

$$MRS_{P;R}=\frac{\sum_{i=1}^{8} nom_i^2}{16}$$
$$=(3^2+3^2+1^2+1^2+3^2+3^2+1^2+1^2)/16$$
$$=40/16$$
$$=2.5$$

$MRS_{P;R}\neq MRS_{Q;R}$, 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 listing 6.7. 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 listing 6.8. 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:

$$DW(\text{“}listing\ 6.8\text{”}) = 4 + 1 + 5 + 2 = 12$$

$$DMR(\text{“}listing\ 6.8\text{”}) = 4/4 = 1$$

$$ME(\text{“}listing\ 6.8\text{”}) = -[(1/4)*\log2(1/4)+(1/4)*\log2(1/4) + (1/4)*\log2(1/4)+ (1/4)*\log2(1/4)] = 2$$

$$MRS(\text{“}listing\ 6.8\text{”}) = (1^2+1^2+1^2+1^2)/4 = 1$$

What we have found for *DW, DMR, ME and MRS* metrics values for the WSDL document given listing 6.7 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

```
…
<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>
…
```

Listing 6.8 Modified part of example WSDLgiven in listing 6.7.

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

Property 9: $(\exists P)\,(\exists Q)\,(\mid P \mid + \mid Q \mid) < (\mid P; Q \mid)$.

From Table 6.1 one can easily see that our measures do not satisfy the original Weyuker's properties. However, as stated in section 4.2.4.1.2 Misra [89] has modifiedthis property and suggest that if $(\exists P)(\exists Q)(|P| + |Q|) \leq (|P;Q|)$, it will be more valuable in evaluating the complexity metric. Therefore, if we refer the same example used in *property 5* and the metrics values for these WSDL documens given in Table 6.1 it can be observed that only *DW* metric satisfies this property.

In this section we have validated our measures through Weyuker's properties. The Table 6.2 shows a summary of evaluation process through Weyuker's properties for our measures. Note that in Table 6.2 satisfied Weyuker's properties are marked.

Table 6.2 Summary of evaluation process for DW, DMR, ME and MRS metrics through Weyuker's properties.

	DW(WSDL)	***DMR***	***ME***	***MRRS***
Property1	✓	✓	✓	✓
Property2	✓	✓	✓	✓
Property3	✓	✓	✓	✓
Property4	✓	✓	✓	✓
Property5	✓	x	✓	✓
Property6	x	✓	✓	✓
Property7	✓	✓	✓	✓
Property8	✓	✓	✓	✓
Property9	✓	x	x	x

6.6 Concluding Remark

As a loosely coupled software components Web services allow integration of heterogeneous systems in diverse domains including business-to business, business-to-consumer and enterprise applications due to the fact that their flexible nature. For providing universal interoperability between these systems flexibility and complexity degree of Web services are main concerns of the Web service developers during

integration process. Particularly, the data flow between interacting applications should be maintained carefully in order to get common agreement on the exchanged data. One of the factors that play a role in the quality of a Web service in terms of maintainability is its data complexity and can be measured by analyzing the messages that are responsible for conveying application data and exchanged by the operations of a Web service. Since the operations offered by a Web service and the messages they exchange are described in WSDL documents, inspecting WSDL documents will help to measure the data complexity of a Web service via the usage of software metrics.

In this chapter we have presented a suite of metrics that focus on the messages describing the arguments of the operations for the assessment of Web service's data complexity. By considering the complexity degree of each argument's data structure which is ignored by the several metrics [68] we have evaluated measures that can provide useful feedback for the developers of Web service in the development life cycle. In order to verify our presented metrics and to prove their usefulness we evaluated them through Weyuker properties as a part of therotical validation process and the result is given in Table 6.2. Additionally, empirical validation has been done and statistics (see Table 1 in Appendix B) that show the comparison results of our metrics with the others are collected through analyzing publicly available WSDL documents written in WSDL 1.1.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this thesis we present a suite of metrics for the schema documents written in W3C XML Schema and DTD, since these are the most favored schema languages for generating XML documents.

The need for the development of these metrics is due to the fact that the eXtensible Markup Language [1] (XML) has become an increasingly significant part of the IT mainstream due to its extraordinary acceptance and popularity. Hence, XML schemas need to be properly designed, so that they can be easily maintained in order for XML data to be effectively and properly used by diverse fields. In terms of maintainability by using the proposed metrics a schema developer can be provided useful feedback about the assessment of the product in the development life cycle. These metrics can enable the quantification of schema size, complexity, quality and the other properties.

All of the newly developed metrics have been demonstrated with examples and supported by comparison with the other well known structure metrics applied on XML Schema documents, DTD and WSDL.

In order 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 properties and examined against the practical framework developed by Kaner [69] as a part of theoretical validation process. A rigorous empirical validation has been done for all proposed metrics except DTD metric. The theoretical, practical and empirical validation proves the worth of the proposed metrics.

Our first metrics *C(XSD)* called Schema complexity metric is based on the internal complexities of each Schema component and is intended to measure physiological complexity of a given Schema document. The usefulness and practical applicability of *C(XSD)* metric has been proven by both theoretical and empirical validations. From empirical validation results it has been shown that *C(XSD)* metric can better capture complexities of XSDs due to internal architectures of Schema components.

Similarly, our second and third metrics, Schema Entropy (SE) and Distinct Structured Elements Repetition Scale (DSERS), are intended to measure

physiological complexity of a given Schema document. These two metrics are based on entropy concept and exploit the directed graph representation of XSDs. They have been verified by both empirical and theoretical validations. These metrics are able to differentiate XSDs in terms of their physiological complexities due to the diversity and repetitions in their elements' structures which has been ignored by existing XSD metrics.

We also propose metrics based on the entropy concept from information theory for the assessment of the structural complexity of XML schema documents written in W3C Document Type Definition (DTD), language.

The XML Web Services, on the other hand, are emerging as the de-facto mechanism for exchanging structured information between applications. Therefore as an XML related technology, we also present a suite of metrics to evaluate and maintain the quality of the XML Web Service. These metrics considering WSDL documents developed for the assessment of XML Web Services are also verified theoretically by using Weyuker's properties and empirical validation of them also shows their usefulness. However, practical evaluation of these metrics through Kaner's framework is aimed as a future work.

7.2 Future Work

As one of our future work, we aim to develop metrics based on the grammatical context of the XML schema documents written in W3C XML Schema and DTD.

Another work to be handled in future can be to evaluate the cognitive complexity of the XML schema documents for XSD and DTD. Since XML has been also using by databases, we are planning to develop a criterion to evaluate and maintain the quality of the XML enabled database in future.

REFERENCES

[1] http://www.w3.org/XML/ *[last visitied 14.09.2008]*

[2] Erl, T.: Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services, Prentice Hall Publischers, 2004.

[3] Cerami, E.: Web Services Essentials, Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. O'Reilly Publischers 2002.

[4] Norman E. Fenton.: Software Metrics – A Rigorous Approach. Chapman &Hall, London, 1991.

[5] http://ivs.cs.uni-magdeburg.de/sw-eng/us/metclas/index.shtml

[6] Shyam R. Chidamber and Chris F. Kemerer.: A Metric Suite for Objectoriented Design. In IEEE Transaction on Software Engineering, volume 20(6), pages 476–493, June 1994.

[7] Horst, Zuse and Karin, Drabe. : Measurement Information System. http://home.t-online.de/home/horst.zuse/zdmis.html, February 2001.

[8] Hogan, Jer.: An Analysis of OO Software Metrics. Technical report, University of Warwick, May 1997.

[9] Anderson, Magnus, Vestergen, Patrik.: Object Oriented Quality Metrics,Master Thesiss, Information Technology Computer Science Dept.Uppsala Univesity,Sweeden,2004

[10] Everald E. Mills.: Software Metrics, SEI Curriculum Module SEI-CM-12-1.1, Software Engineering Institute, Carnegie Mellon University,1988

[11] Sandeep, Purao and Vaishnavi, Vijay.: Product Metrics for Object-Oriented Systems, ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 191–221.

[12] Choi, Byron. : A Few Tips for Good XML Design. Technical report, University of Pennsylvania, http://db.cis.upenn.edu/˜kkchoi/DTDI2/, November 2000.

[13] Sahuguet, Arnaud. : Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask. In Third International Workshop WEBDB2000, volume 1997 of Lecture Notes in Computer Sience, pages 171–183. Springer, May2000.

[14] Z. Lin, B. He and B. Choi: A Quantitative Summary of XML Structures. ER 2006. Page 228-240. LNCS 4215, Springer-Verlag.

[15] Qureshi, Mustafa H., Smadzadeh, M. H.: Determining the Complexity of XML Documents, Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II - Volume 02, pp. 416 – 421,April 2005.

[16] McDowell, A., Schmidt, C., Yue, K.:Analysis and Metrics of XML Schema. In SERP '04, Proceedings of the International Conference on Software Engineering Research and Practice, 538-544. CSREA Press(2004)

[17] R. L¨ammel, R., Kitsis, S., Remy, D.: Analysis of XML schema usage. In Conference Proceedings XML 2005(2005)

[18] Visser,J.:Structure Metrics for XML Schema, Proceedings of XATA.(2006)

[19] Klettke, M., Scneider,L.,Heuer, A.: Metrics for XML Document Collections, XMLDM Workshop,Czech Republic,2002,pp.162-176
[20] The Whole Internet User's Guide & Catalog, by Ed Krol, was published in September 1992 by Oreilly
[21] http://www.isoc.org/internet/history/brief.shtml]
[22] Dick, Kevin.: XML: A Manager's Guide, Second Edition ,Addison Wesley Professional, August 28, 2002
[23] Morrison, Michesal. : XML Unleashed, Sams Publication, December 21, 1999
[24] http://www.luminoussolutions.com/data/history_of_markup.pdf
[25] java.sun.com/dtd/ *[last visitied 14.09.2008]*
[26] http://struts.apache.org/dtds/ *[last visitied 14.09.2008]*
[27] http://jonas.objectweb.org/dtds/ *[last visitied 14.09.2008]*
[28] http://www.ncbi.nlm.nih.gov/dtd/ *[last visitied 14.09.2008]*
[29] http://www.cs.helsinki.fi/group/doremi/publications/XMLSCA2000.html *[last visitied 14.09.2008]*
[30] http://www.pramati.com/dtd/ *[last visitied 14.09.2008]*
[31] Binstock, C., Peterson, D., Smith, M., Wooding, M., Dix, C., Galtenberg, C.: The XML Schema Complete Reference, Addison Wesley Professional Publishers.(2002)
[32] http://www.stylusstudio.com/xml_schema_doc_gen.html
[33] http://www.w3.org/TR/REC-xml-names/
[34] http://www.w3.org/TR/xslt20/
[35] http://www.w3.org/TR/xquery/
[36] http://www.w3.org/TR/xpath20/
[37] http://www.w3.org/TR/1998/REC-xml-19980210
[38] http://www.xfront.com/GlobalVersusLocal.html; http://www.oreillynet.com/xml/blog/2006/05/metrics_for_xml_projects_1_ele .html
[39] Van der Vlist, Eric. : XML Schema. O'Reilly Publication, 2002.
[40] https://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=/com .ibm.db2.luw.xml.doc/doc/c0051286.html
[41] Briand, L.C., Morasca, S., Basily. : Property based Software Engineering Measurement. IEEE Transactions on SE, vol. 22, 1, pp.68-86, 1996.
[42] http://www.w3.org/TR/wsdl http://www.w3.org/TR/wsdl20/
[43] Boxall, M. and S. Araban, S.: Interface Metrics for Reusability Analysis of Components, in ASWEC '04: Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04), 2004.
[44] Davis, J., & LeBlanc, R.: A Study of the Applicability of Complexity Measures. IEEE Transactions on Software Engineering, 1998.
[45] Harrison, W.: An entropy-based Measure of Software Complexity, IEEE Transactions on Software Engineering, (1992), 18, 1025.1029.
[46] Torres, W. R., & Samadzadeh, M. H.: Software Reuse and Information Theory Based Metrics, IEEE Transactions on Software Engineering, 1990.
[47] Mohanty, S. N.: Entropy metrics for software design evaluation, The Journal of Systems and Software, 2, 39.46,(1981).
[48] K. Kim, Y. Shin, C. Wu, "Complexity Measures for Object-Oriented Program Based on the Entropy," *apsec*, p. 127, Second Asia-Pacific Software Engineering Conference (APSEC'95), 1995

[49] Edward B. Allen., Taghi M. Khoshgoftaar., Ye, Chen.: Measuring Coupling and Cohesion of Software Modules: An Information-Theory Approach," *metrics*, p. 124, Seventh International Software Metrics Symposium (METRICS'01), 2001

[50] Edward B. Allen., Sampath, Gottipati., Rajiv, Govindarajan.: Measuring size, complexity, and coupling of hypergraph abstractions of software: An information-theory approach, Software Quality Control, v.15 n.2, pp.179-212, June 2007

[51] Salwa K. Abd-El-Hafiz.: Entropies as Measures of Software Information, 17th IEEE International Conference on Software Maintenance (ICSM'01), 2001

[52] Shannon, C.E., and Weaver, W.: The Mathematical Theory of Communication, Urbana, IL: University of Illinois Press, 1949.

[53] Chapin, N.: An entropy metric for software maintainability System Sciences, Proceedings of the Twenty-Second Annual Hawaii International Conference Vol.II: Software Track , 3-6 Jan 1989.

[54] Gaffney, Jhon. : Instruction entropy, a possible measure of program/architecture compatibility, ACM SIGMETRICS Performance Evaluation Review, Volume 12 Issue 4, publisher: ACM, December 1984.

[55] Etzkorn, Letha, Gholston, Sampson, and Hughes, William E., Jr.: A Semantic Entropy Metric, Journal of Software Maintenance and Evolution, Vol. 14, Issue 4, July/August, 2002, pp. 293-310.

[56] Yuming Zhou, Baowen Xu.: Measuring structural complexity for class diagrams: an information theory approach Symposium on Applied Computing Proceedings of the 2005 ACM symposium on Applied computing SAC'05, pp.1679 – 1683.

[57] Bansiya, Jagdish, Davis, Carl, and Etzkorn, Letha.: An Entropy Based Complexity Measure for Object-Oriented Designs, Theory and Practice of Object Systems, Vol. 5, No. 2, May, 1999, pp.1-9.

[58] Zweben, S., Halstead, M.: The Frequency Distribution of Operators in PL/1 programs, IEEE Trans on S/W Eng., Vol.SE-3, No.2, pp91-95, March 1979.

[59] Gourley, D., Totty, B.: HTTP: The Definitive Guide. O'Reilly Publischers, 2002.

[60] Newcomer, E.,Greg Lomow,G.: Understanding SOA with Web Services, Addison Wesley Professional, 2004.

[61] 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.

[62] http://docbook.sourceforge.net/release/dsssl/current/dtds/

[63] WS-I Basic Profile Version 1.0
http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

[64] http://www.w3.org/Math/XMLSchema/mathml2/content/common-attrib.xsd

[65] ISO/IEC:Information Technology – Text and Office Systems – Regular Language
Description for XML (RELAX) – Part 1: RELAX Core, 2000.D TR 22250-

[66] http://www.xml.gr.jp/relax. *[last visitied 14.09.2008]*
http://www.relaxng.org/

[67] Basci, D., Misra,S.: Complexity Metric for XML Schema Documents, Proceeding of 5th international Workshop on SOA and Web Services,OOPSLA2007

[68] Yu, Y., Lu, J., Fernandez-Ramil, J., Yuan,P.: Comparing Web Services with other Software Components, Proceeding of IEEE International Conference on Web Services ,ICWS 007, 2007.

[69] Kaner, C.: Software Engineering Metrics: What do they Measure and how do we know? , Proceedings of 10th International Software Metrics Symposium, Metrics 2004.

[70] http://www.w3.org/TR/2004/WD-wsdl20-20040803/#migration

[71] McCabe, T.J: A complexity measure, IEEE Trans. Software Eng.,vol. SE-2, pp. 308-320, 1976.

[72] Henry, S. and Kafura, D. : The Evaluation of Software Systems' Structure Using Quantitative Software Metrics, Software Practice and Experience, pp. 561-573, June 1984.

[73] Weyuker, E.: Evaluating Software Complexity Measures. IEEE Transactions on Software Engineering, vol. 14, pp.1357-1365, 1998.

[74] http://www.w3.org/TR/soap/ *[last visitied 14.09.2008]*

[75] http://www.oasis-open.org/home/index.php *[last visitied 14.09.2008]*

[76] http://www.oasis-open.org/specs/index.php#uddiv2 *[last visitied 14.09.2008]*
http://xml.coverpages.org/uddi.html
http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

[77] http://www.w3.org/Protocols/ *[last visitied 14.09.2008]*

[78] http://www.w3.org/2002/ws/ *[last visitied 14.09.2008]*

[79] http://www.research.ibm.com/journal/sj/412/gottschalk.html

[80] Aleyne Lapeyre, Deborah. : Introduction to XML Schema Languages, Mulberry Technologies, Inc.
http://www.mulberrytech.com
http://xml.coverpages.org/schemas.html

[81] http://xml.ascc.net/resource/schematron/schematron.html

[82] http://www.w3.org/TR/NOTE-SOX/

[83] Van der Vlist, Eric.:XML Schema Languages Compared, XML Prague 2005 May 2005, http://dyomedea.com/papers/2005-xmlprague

[84] Martens, Wim., and Neven, Frank.: Expressiveness and Complexity of XML Schema, ACM Transactions on Database Systems, Vol. 31, No. 3, pp.770-813, September 2006.

[85] Dam Nielsen, Janus.: Relations Between Schema Languages for XML, Master Thesis, February 24,2006

[86] Fenton, N.E.: When a software measure is not a measure. Software Engineering Journal, 1992, pp. 357-362, 1992.

[87] Misra S. and Kilic, H.: Measurement theory and Validation Criteria for Software Complexity measure. ACM SIGSOFT SEN. 31, no.6, pp.1-3, 2006

[88] IEEE Computer Society: Standard for software Quality Metrics Methodology. Revision IEEE Standard (1998) 1061-1998.

[89] Misra, Sanjay: Modified Set of Weyuker's Properties, 5th IEEE Int. Conf. on Cognitive Informatics (ICCI'06), 2006

[90] http://www.fef.gazi.edu.tr/turkish/statist/dokuman/T_Table.pdf
http://www.akademikdestek.net/kutuphane/analiz/analiz_dosyalar/parametrik_nonparametrik_testler.doc

[91] Uğur, A.Naci : Kalite Geliştirme İçin İstatistik Yöntemler,2000

[92] http://www.omegahat.org/XML/DTDs/ *[last visited:15.09.2008]*
http://www.openmobilealliance.org/Technical/dtd.aspx
http://fisheye5.cenqua.com/browse/glassfish/update-center/dtds/
http://www.python.org/topics/xml/dtds/
http://www.okiproject.org/polyphony/docs/raw/dtds/

APPENDICES

APPENDIX A

```
<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>
```

```
      </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="GetMessageByIDSoapIn"><!--G4(2,1)-->
  <wsdl:part element="tns:GetMessageByID" name="parameters"/>
 </wsdl:message>
 <wsdl:message name="GetMessageByIDSoapOut"><!--G5(13,9)-->
  <wsdl:part element="tns:GetMessageByIDResponse" name="parameters"/>
 </wsdl:message>
 <wsdl:message name="GetMessageByIDMessengerHeader"><!--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"/>
  <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="literal"/>
    </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>
```

Listing 1 Real life example WSDL documents referenced from chapter 6.

APPENDIX B

Table 1 The values of the metrics presented in chapter 6 for WSDL documents. The references for these documents are given in Table 2 in Appendix C.

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*

Table 1 (cont.)

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	a8	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

APPENDIX C

Table 2 Web links for XSD and WSDL documents referenced from chapter 4 and chapter 6 respectively.

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*

Table 2(cont.)

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
57	www.banguat.gob.gt/variables/ws/BDEF.asmx?WSDL
58	dev.w3.org/2006/xquery-test-suite/TestSuiteStagingArea/XQTSCatalog.xsd
59	www.oasis-open.org/committees/regrep/documents/2.0/schema/rim.xsd
60	bible.sumerano.com/bible.asmx?WSDL
61	www.vbcentral.net/ShipService/Services.asmx?WSDL
62	www.xpyder.co.kr/XpyWebService/member.asmx
63	www.xignite.com/xNews.asmx?WSDL
64	http://hooch.cis.gsu.edu/bgates/MathStuff/Mathservice.asmx?WSDL
65	www.esendex.com/secure/messenger/soap/ContactService.asmx?WSDL
66	mlbs.net/nacgeoservices/geoservices.asmx?WSDL
67	www.codeplex.com/ajaxdoc/SourceControl/FileView.aspx?itemId=102696&changeSetId=716
68	www.oasis-open.org/committees/regrep/documents/2.0/schema/query.xsd
69	www.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL
70	ws.strikeiron.com/GaleGroupBusinessInformation?WSDL
71	soap.einsteinware.com/nascar/nascardataservice.asmx?WSDL
72	ws.netedgesoftware.com/wsenabler/1.0/StockInfoCS.asmx?WSDL
73	www.golemproject.com/Apps/96/Generator.asmx?WSDL
74	services.argosoft.com/AddressValidation/AddressVerifier.asmx?WSDL
75	www.mathertel.de/AJAXEngine/S02_AJAXCoreSamples/CalcService.asmx?WSDL
76	trial.serviceobjects.com/pa/phoneappend.asmx?WSDL
77	del.eterio.us/blog/editposts.asmx?WSDL

Table 2(cont.)

78	ws.cisa.ca/WehireWS/JobsWs.asmx?WSDL
79	api.legiomedia.com/Content.asmx?WSDL
80	www.sipeaa.it/wset/ServiceET.asmx?WSDL
81	www.xignite.com/xwatchlists.asmx?WSDL
82	service.test.cdream.com.cn/CernetForSP/CernetInterfaceForSP.asmx?WSDL