

# **XML Grammar and Parser for the Web Service Offerings Language**

by

**Kruti Patel, B. Eng.**

A thesis submitted to the Faculty of Graduate Studies and Research  
in partial fulfillment of the requirements for the degree of

**Master of Applied Science  
in Electrical Engineering**

Ottawa-Carleton Institute for Electrical and Computer Engineering  
Faculty of Engineering  
Department of Systems and Computer Engineering  
Carleton University,  
Ottawa, Ontario, Canada, K1S 5B6

January 28, 2003

© 2003, Kruti Patel

The undersigned hereby recommend to  
The Faculty of Graduate Studies and Research  
acceptance of the thesis,

## XML Grammar and Parser for the Web Service Offerings Language

submitted by

Kruti Patel,

In partial fulfillment of the requirements  
for the degree of Master of Applied Science

---

Professor Bernard Pagurek, Thesis Supervisor

---

Chairman, Department of Systems and Computer Engineering

Carleton University

January 28, 2003

## **Abstract**

---

A Web Service is a service that can be accessed over a network using XML based messages communicated through Internet protocols. The Web Service Description Language (WSDL) enables description of the operations supported by a Web Service, the location of a Web Service, and methods for accessing a Web Service in XML. WSOL (Web Service Offerings Language) is a novel XML-based language, compatible with WSDL 1.1, for formal specification of constraints, management statements, multiple classes of service for Web Services and contracts between Web Services. Explicit and precise specification of such information is essential for management of Web Services and Web Service compositions. The focus of this thesis is on the development of XML grammar and parser for WSOL. The WSOL grammar is developed using the XML Schema language. The WSOL grammar enables generation of WSOL documents that conform to the grammar. A WSOL parser named “Premier” is developed in Java, based on the Apache Xerces2 Java XML parser. The WSOL parser enables validation of WSOL and WSOL-related documents and detection and notification of various syntax and semantic errors in them. An illustrative WSOL example is also developed that enables verification of the developed WSOL grammar and the WSOL parser.

## **Acknowledgements**

---

I am very thankful to my thesis supervisor, Professor Bernard Pagurek, for his guidance, support, patience and encouragement throughout my research work. I am also very grateful to Vladimir Tosic (a Ph.D. candidate in the Network Management Lab, Carleton University) for always guiding me in the right direction, understanding me, being patient with me, motivating me and making me believe in myself, during my research. I would like to express my thanks to all my colleagues in the Network Management Lab, for their help. Finally, I would like to thank God, my family and my friends for always being by my side throughout my studies, for loving me unconditionally, for understanding me, and for constantly inspiring me to reach my goals.

# Table of Contents

---

<b>ABSTRACT .....</b>	<b>III</b>
<b>ACKNOWLEDGEMENTS.....</b>	<b>IV</b>
<b>TABLE OF CONTENTS.....</b>	<b>V</b>
<b>LIST OF FIGURES .....</b>	<b>VIII</b>
<b>LIST OF TABLES .....</b>	<b>X</b>
<b>1 INTRODUCTION.....</b>	<b>1</b>
1.1 BACKGROUND .....	1
1.2 MOTIVATION .....	3
1.3 THESIS GOALS .....	5
1.4 THESIS CONTRIBUTION.....	5
1.5 THESIS ORGANIZATION .....	6
<b>2 OVERVIEW OF THE WEB SERVICES ARCHITECTURE, TECHNOLOGIES, AND XML PARSING.....</b>	<b>7</b>
2.1 WEB SERVICES ARCHITECTURE.....	7
2.2 WEB SERVICE TECHNOLOGIES.....	9
2.2.1 <i>SOAP</i> .....	10
2.2.2 <i>WSDL</i> .....	14
2.3 XML PARSING.....	25
2.4 XERCES2 JAVA PARSER.....	27
2.5 SUMMARY .....	31
<b>3 WEB SERVICE OFFERINGS LANGUAGE (WSOL) AND ITS RELATED WORKS .....</b>	<b>32</b>
3.1 WSOL.....	33
3.1.1 <i>Constraint</i> .....	33
3.1.2 <i>Statement</i> .....	35

3.1.3	<i>Constraint Group (CG), Constraint Group Template (CGT), Service Offering</i> .....	38
3.1.4	<i>Benefits of WSOL</i> .....	41
3.2	RELATED WORKS.....	43
3.2.1	<i>Works Most Relevant to WSOL</i> .....	44
3.2.1.1	WSLA.....	44
3.2.1.2	WSML.....	46
3.2.1.3	DAML-S.....	48
3.2.1.4	WSEL.....	52
3.2.2	<i>Works Less Relevant to WSOL</i> .....	53
3.2.2.1	CPP and CPA concepts of ebXML.....	53
3.2.2.2	XL.....	54
3.2.2.3	Other Works.....	57
3.2.3	<i>Works Least Relevant to WSOL</i> .....	58
3.2.3.1	QML.....	58
3.2.3.2	DBC.....	58
3.3	SUMMARY.....	59
<b>4</b>	<b>WSOL GRAMMAR.....</b>	<b>64</b>
4.1	WSOL SCHEMA.....	65
4.1.1	<i>Grammar for the WSOL root element</i> .....	66
4.1.2	<i>Grammar for a WSOL Constraint</i> .....	68
4.1.3	<i>Grammar for a WSOL Statement</i> .....	70
4.1.3.1	<i>Grammar for a WSOL Subscription Statement</i> .....	71
4.1.3.2	<i>Grammar for a WSOL Price Statement</i> .....	73
4.1.3.3	<i>Grammar for a WSOL Price Default Statement</i> .....	74
4.1.3.4	<i>Grammar for a WSOL External Operation Call Statement</i> .....	75
4.1.3.5	<i>Grammar for a WSOL Management Responsibility Statement</i> .....	77
4.1.3.6	<i>Grammar for a WSOL Include Statement</i> .....	79
4.1.3.7	<i>Grammar for a WSOL Instantiate Statement</i> .....	80
4.1.4	<i>Grammar for a WSOL Constraint Group (CG)</i> .....	82
4.1.5	<i>Grammar for a WSOL Constraint Group Template (CGT)</i> .....	85
4.1.6	<i>Grammar for a WSOL Service Offering</i> .....	86
4.2	GRAMMAR FOR THE SPECIFICATION OF DYNAMIC RELATIONSHIPS BETWEEN SERVICE OFFERINGS 94	
4.3	SUMMARY.....	96
<b>5</b>	<b>THE WSOL PARSER.....</b>	<b>97</b>
5.1	DESIGN OF THE WSOL PARSER.....	97
5.2	MAIN FEATURES OF THE WSOL PARSER.....	107
5.3	SUMMARY.....	112
<b>6</b>	<b>DEMONSTRATION OF THE WSOL PARSER.....</b>	<b>113</b>
6.1	AN EXAMPLE OF CORRECT PARSING.....	115
6.2	EXAMPLES OF DISCOVERED SYNTAX AND SEMANTIC ERRORS.....	118
6.2.1	<i>Errors in Constraints</i> .....	118
6.2.2	<i>Errors in Subscription Statements</i> .....	121
6.2.3	<i>Errors in Price Default/Penalty Default Statements</i> .....	122
6.2.4	<i>Errors in Price/Penalty Statements</i> .....	122
6.2.5	<i>Errors in Management Responsibility Statements</i> .....	123
6.2.6	<i>Errors in External Operation Call Statements</i> .....	124
6.2.7	<i>Errors in Include Statements</i> .....	125
6.2.8	<i>Errors in CGT Instantiation Statements</i> .....	126

6.2.9	<i>Errors in CGs</i> .....	127
6.2.10	<i>Errors in CGTs</i> .....	128
6.2.11	<i>Errors in Service Offerings (SOs)</i> .....	129
6.3	SUMMARY.....	131
<b>7</b>	<b>CONCLUSIONS, SUMMARY OF CONTRIBUTIONS AND FUTURE WORK.....</b>	<b>132</b>
7.1	CONCLUSIONS.....	132
7.2	SUMMARY OF CONTRIBUTIONS.....	133
7.3	FUTURE WORK.....	135
<b>8</b>	<b>REFERENCES.....</b>	<b>137</b>
	<b>APPENDIX A: WSOL SCHEMA.....</b>	<b>145</b>
	<b>APPENDIX B: EXPRESSION SCHEMA.....</b>	<b>152</b>
	<b>APPENDIX C: BNF NOTATION FOR THE GRAMMAR SPECIFIED IN THE EXPRESSION SCHEMA.....</b>	<b>162</b>
	<b>APPENDIX D: WSDL DESCRIPTION FOR THE BUYSTOCK WEB SERVICE EXAMPLE.....</b>	<b>164</b>
	<b>APPENDIX E: WSOL DESCRIPTION FOR THE BUYSTOCK WEB SERVICE EXAMPLE.....</b>	<b>167</b>
	<b>APPENDIX F: “QOSMEASONTOLOGY.ONT” (AN ONTOLOGY OF QOS MEASUREMENT UNITS).....</b>	<b>174</b>

## List of Figures

---

Figure 2.1 Web Services Architecture .....	7
Figure 2.2 Web Services Protocol Stack .....	8
Figure 2.3 Structure of a SOAP message .....	11
Figure 2.4 Example of a SOAP Request Message .....	13
Figure 2.5 Client accessing a Web Service described using WSDL.....	14
Figure 2.6 Overview of XML Parsing.....	26
Figure 2.7 Design details of Xerces2 Java Parser .....	28
Figure 4.1 Grammar for the WSOL root element .....	66
Figure 4.2 Graphical representation of the grammar for the WSOL root element .....	67
Figure 4.3 Example of a WSOL root element .....	68
Figure 4.4 Grammar for a WSOL Constraint .....	69
Figure 4.5 Example of a WSOL Constraint.....	70
Figure 4.6 Grammar for a WSOL Subscription Statement .....	71
Figure 4.7 Example of a WSOL Subscription Statement .....	72
Figure 4.8 Grammar for a WSOL Price Statement .....	73
Figure 4.9 Example of a WSOL Price Statement .....	73
Figure 4.10 Grammar for a WSOL Price Default Statement.....	74
Figure 4.11 Example of a WSOL Price Default Statement .....	75
Figure 4.12 Grammar for a WSOL External Operation Call Statement .....	75
Figure 4.13 Example of a WSOL External Operation Call Statement.....	76
Figure 4.14 Grammar for a WSOL Management Responsibility Statement.....	77
Figure 4.15 Example of a WSOL Management Responsibility Statement .....	78
Figure 4.16 Grammar for a WSOL Include Statement .....	79
Figure 4.17 Example of a WSOL Include Statement.....	79
Figure 4.18 Grammar for a WSOL Instantiate Statement .....	80
Figure 4.19 Example of a WSOL Instantiate Statement .....	81
Figure 4.20 Grammar for a general WSOL Statement.....	81
Figure 4.21 Grammar for a WSOL Constraint Group (CG).....	82
Figure 4.22 Graphical representation of the grammar for a WSOL constraint group .....	83
Figure 4.23 Example of a WSOL constraint group.....	84
Figure 4.24 Example of a WSOL constraint group template .....	85
Figure 4.25 Grammar for a WSOL Service Offering.....	86
Figure 4.26 Graphical representation of the grammar for a WSOL service offering .....	87
Figure 4.27 Example of a WSOL service offering .....	89
Figure 4.28 Grammar for the WSOL import mechanism.....	89
Figure 4.29 Grammar for the Specification of Dynamic Relationships between Service Offerings.....	95
Figure 4.30 Example of a specification of Dynamic Relationships between Service Offerings.....	96
Figure 5.1 Class Diagram of the WSOL Parser .....	99
Figure 5.2 Sequence Diagram of the WSOL Parser.....	100
Figure 5.3 Example of scoping in symbol tables .....	106



Figure 5.4 Overview of how a WSOL Parser works.....	109
Figure 6.1 A syntactically and semantically correct segment of the WSOL document..	115
Figure 6.2 DOM tree of the parsed WSOL document .....	116
Figure 6.3 Symbol Tables generated for the parsed WSOL document.....	117
Figure 6.4 An error detected in a constraint .....	118
Figure 6.5 An error detected in a constraint .....	119
Figure 6.6 An error detected in a constraint .....	120
Figure 6.7 An error detected in a constraint .....	121
Figure 6.8 An error detected in a subscription statement .....	121
Figure 6.9 An error detected in a price default statement .....	122
Figure 6.10 An error detected in a price statement .....	123
Figure 6.11 An error detected in a management responsibility statement .....	123
Figure 6.12 An error detected in a management responsibility statement .....	124
Figure 6.13 An error detected in an external operation call statement .....	125
Figure 6.14 An error detected in an Include statement .....	126
Figure 6.15 An error detected in a CGT Instantiation statement.....	127
Figure 6.16 An error detected in a CG .....	127
Figure 6.17 An error detected in a CG .....	128
Figure 6.18 An error detected in a CGT.....	128
Figure 6.19 An error detected in a CGT.....	129
Figure 6.20 An error detected in a service offering .....	130
Figure 6.21 An error detected in a service offering .....	130
Figure 6.22 An error detected in a service offering .....	131

## List of Tables

---

Table 3.1 Comparison between WSOL and the works most relevant to WSOL.....	61
--	----

# 1 Introduction

---

## 1.1 Background

A Web Service [9, 53] is any service (i.e., a software application) that can be accessed over a network using XML based messages communicated through Internet protocols. The interfaces and bindings supported by a Web Service are defined and described using XML, and can be discovered. A Web Service is platform-independent and programming language-independent.

Examples of Web Services [5] are: reporting current weather information of a particular city, providing latest stock quotes, providing latest news, reporting exchange rate between two currencies, providing dictionary meanings for a given word, etc.

A Web Service (its functionality, location, and access methods) can be described in XML using WSDL (Web Services Description Language). But, the WSDL language does not enable specification of constraints, management statements, classes of service for Web Services and contracts between Web Services. Explicit and precise specification of such information is essential for management of Web Services and Web Service compositions. For example, specification of functional constraints, such as pre-conditions and post-conditions, useful in determining whether a Web Service behaves correctly or not, is important for fault management. Similarly, specification of QoS constraints, useful in monitoring and metering of QoS metrics for Web Services, is essential for performance management. Specification of price and penalty statements for Web Services is vital for accounting management. Also, specification of classes of service for

Web Services and contracts between Web Services, that contain groups of various constraints and management statements, is useful in handling the complexity of management information in a better way, and therefore, it is important for all management activities.

A new language called WSOL (Web Service Offerings language) [44, 45, 50], fully compatible with WSDL, is being developed in our research group at the Network Management Lab (Carleton University) by Vladimir Tomic, a Ph.D. candidate, to enable such specification in XML. The main goals of WSOL are as follows:

1. To be usable for management of Web Services. That is for monitoring, measurement of QoS metrics, evaluation of various constraints, accounting, and billing.
2. To be usable for management of Web Service compositions using dynamic adaptation mechanisms such as switching between service offerings, deactivation/reactivation of service offerings, and creation of new appropriate service offerings.
3. To be usable for more precise selection of appropriate Web Services and their service offerings, e.g., in the process of dynamic Web Service composition. WSOL supports this goal by enabling comprehensive specification of Web Services, service offerings, and static and dynamic relationships between service offerings.
4. To have the expressive power to enable reuse of specifications.
5. To be able to reduce run-time overhead incurred while monitoring and management without disabling management activities.

6. To be able to reduce unexpected interactions between the composed Web Services by enabling comprehensive formal specification of Web Services.
7. To be fully compatible with WSDL, a standard language for the specification of functionality, access methods, and location of Web Services. WSOL supports this goal by enabling reuse of information specified using WSDL and additional specification of various constraints, management statements, and classes of service for Web Services without modifying the referenced WSDL information.

## **1.2 Motivation**

When I started working on my thesis, WSOL was just an idea, a set of vague concepts and requirements. It did not have any grammar and, hence, it was not yet a language. Since there was no WSOL grammar, it was not possible to create WSOL documents conforming to the grammar. Without a precise grammar, the WSOL language could not be used. Consequently, there was a strong need for defining the grammar for the WSOL language in XML. The grammar for the WSOL language had to be specified in XML since there was no other choice for Web Services. Another reason for doing so was that WSOL had to be compatible with the WSDL language that is XML-based. Additionally, expressing the grammar for the WSOL language in XML provides several benefits. XML enables the WSOL grammar to specify the meaning and the structure of the WSOL language concepts. XML, being a platform-independent as well as a programming language independent language, enables interoperability between diverse

applications more easily. Moreover, specifying the WSOL grammar in XML enables generation of the WSOL parser more easily. This is because a number of parsers, such as Xerces2, are already available for XML. These XML parsers can be referred to and/or reused for creating the WSOL parser.

The WSOL language also did not have any tool for validation of WSOL documents and detection of eventual syntax and semantic errors in them. Therefore, the WSOL language needed a parser to validate the WSOL documents against the WSOL grammar and to detect syntax and semantic errors in them. Among the tools that can be developed for the WSOL language, a parser is one of the basic tools and a very important tool. Successful development of a parser verifies that the suggested grammar for a language is syntactically correct and implementable by tools such as parsers and, compilers. Further, it can verify whether the WSOL documents written according to the WSOL grammar are syntactically and semantically correct or not. It can also detect and report syntax and semantic errors if any occurs. If any error is detected, the parser can discover and notify the location in the WSOL document where the error occurred. Moreover, the parser can be used to create other tools (such as automatic code generator, compiler, etc.) for the WSOL language. Additionally, developing the parser can help in improving the WSOL language and the WSOL grammar because it enables better understanding of language-related issues. Looking at the benefits a parser would provide, it seems very important to develop a parser for the WSOL language.

### **1.3 Thesis Goals**

The two main goals of this Master's thesis were:

1. To develop an XML grammar for the WSOL language using the XML Schema language.
2. To develop a corresponding parser for the WSOL language. This parser should discover syntax and semantic errors in WSOL documents and be reusable for a future WSOL compiler.

### **1.4 Thesis Contribution**

The main contributions of this Master's thesis are:

1. The XML grammar for the WSOL language was developed using the XML Schema language. This grammar verifies implementability of the WSOL language architecture and constructs. The grammar also enables creation of WSOL documents conforming to the grammar.
2. A parser for the WSOL language, named "Premier" WSOL parser, was developed. This parser verifies that the XML grammar for WSOL is well-formed and syntactically correct. The parser validates the WSOL documents and detects and reports syntax and semantic errors in them.

3. An illustrative WSOL example was developed that verifies the developed WSOL grammar and the WSOL parser.
4. Based on the experiences with the WSOL grammar and the WSOL parser, feedbacks and ideas were provided for the improvement of the WSOL language.

## **1.5 Thesis Organization**

The rest of the thesis is organized as follows. Chapter 2 provides an overview of the basic Web Services Architecture and its main components. This Chapter also discusses several Web Services technologies including WSDL. A brief review of XML parsing and the Xerces2 Java XML parser is also provided in this chapter. Chapter 3 discusses the WSOL language in detail. This chapter also briefly discusses the main concepts of WSOL-related languages, their tools, and ways in which they differ from WSOL. In Chapter 4, only significant parts of the developed WSOL grammar are described in detail. Chapter 5 provides the design details of the developed WSOL parser. This chapter also discusses the main features of the WSOL parser. Chapter 6 demonstrates the ability of the WSOL parser to detect and report syntax and semantic errors using an example. Finally, in Chapter 7, conclusions and contributions of this thesis are given, followed by suggestions for related future work.



## 2 Overview of the Web Services Architecture, Technologies, and XML Parsing

---

This chapter discusses a basic Web Services Architecture and its main components. The chapter also provides an overview of several Web Service technologies with detailed discussions of Web Service technologies relevant to the thesis. A brief review of parsing XML documents is also provided in this chapter. A number of XML parsers exist, but, only the Xerces2 Java XML parser is discussed here since it is relevant to the thesis.

### 2.1 Web Services Architecture

The Web Services architecture [9, 19, 26], as shown in Figure 2.1, consists of three main roles: the service provider, the service consumer, and the service registry. A service provider implements the Web Service(s) it offers and publishes them with the service registry. A service consumer finds a Web Service it needs by searching the service registry and invokes that Web Service. A service consumer can either be a human being, or another Web Service.

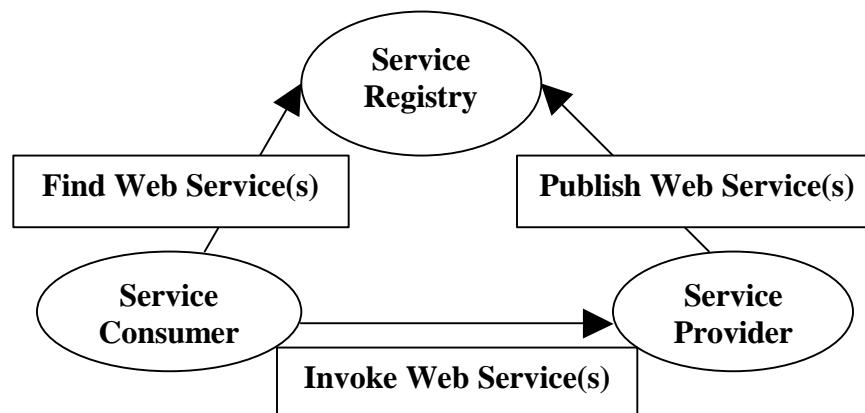
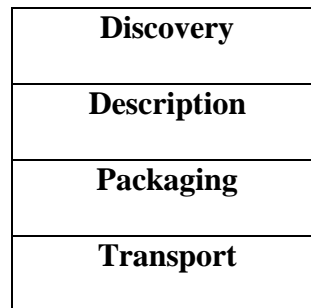


Figure 2.1 Web Services Architecture

The Web Services protocol stack implements the Web Services architecture (i.e., each layer in the protocol stack provides some functionality with associated technologies or protocols that enable the actors in the Web Services architecture to perform their specific operations). The general Web Services protocol stack, as shown in Figure 2.2 [9, 26], consists of four main layers: Transport, Packaging, Description, and Discovery.



**Figure 2.2 Web Services Protocol Stack**

The transport layer enables communication between the Web Service providers, consumers, and registries by transporting the XML messages between them.

The packaging layer, also known as the XML messaging layer, packages the messages being exchanged, in a common XML format. This XML format can be understood by all communicating parties and if present, the intermediaries as well.

The description layer enables the service provider to describe the Web Services it provides i.e., to describe the main functionality offered by the Web Services, the location of the Web Services, and the transport protocol, XML messaging protocol, etc. supported by the Web Services.

The discovery layer enables the service provider to publish the Web Services it offers with a service registry. It also enables the service consumers to discover the Web

Services offered by the service providers and to obtain the descriptions of the discovered Web Services.

## **2.2 Web Service Technologies**

Each layer in the Web Services protocol stack is associated with one or more Web Service technologies or protocols that enable the corresponding layer to provide its functionality.

The transport layer in the Web Services protocol stack is associated with a number of protocols such as Hyper Text Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), etc. Any one of these transport protocols can be supported by a Web Service for communication between itself and the service consumer. Currently, the transport protocol most commonly supported by Web Services is HTTP since it is a simple, stable, and a widely deployed transport protocol [9].

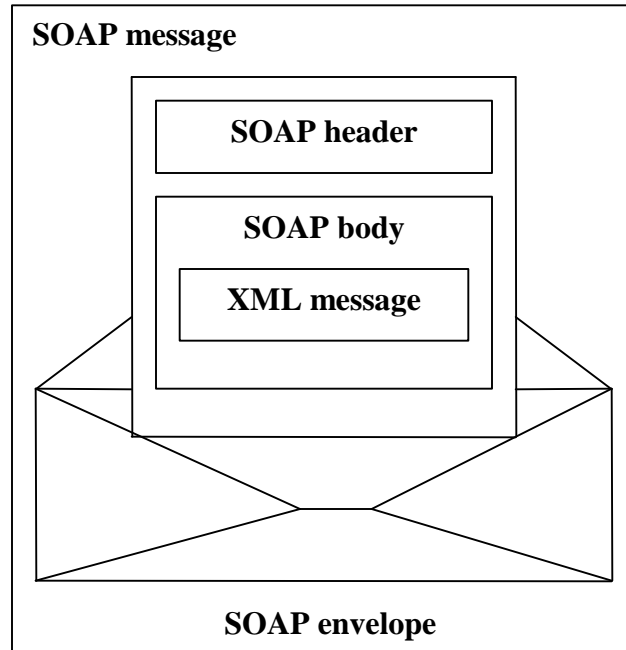
The packaging layer in the Web Services protocol stack is associated with packaging protocols such as SOAP, XML-RPC, etc. These packaging protocols are based on the XML language [8, 27, 31] for several reasons, as mentioned in [9, 26]:

- XML enables specification of the meaning and structure of the data to be transferred.
- Also, a number of XML tools, such as XML parsers and XML editors, are easily available today for almost every programming language.
- XML is platform-independent as well as programming language independent. Hence, it enables sharing of data between diverse computer systems and diverse applications more easily.

### 2.2.1 SOAP

The SOAP packaging protocol [9, 12, 23, 26] is a standardized protocol that defines the structure of the SOAP message for packaging the XML messages to be transferred over a network between applications and provides a set of encoding rules for representing the application and platform-specific data types in XML syntax. The SOAP protocol has two main applications: Remote Procedure Calls (RPCs) and Electronic Document Interchange (EDI). In the case of Remote Procedure Calls, the SOAP request message consists of the remote method name and the method's parameters while the SOAP response message consists of the remote method's return values. In the case of Electronic Document Interchange, the SOAP message consists of an entire XML document or a part of it representing the information that needs to be transferred e.g. product information.

The SOAP message structure, as shown in Figure 2.3 [9, 26], consists of an outer envelope called the SOAP envelope. The SOAP envelope consists of a header called the SOAP header and a body called the SOAP body. The SOAP header is optional while the SOAP body is required. The SOAP header provides some information regarding the processing of the XML message by the receiving application or by the intermediaries in the SOAP message path. This information can contain unique identifiers of the intermediaries, routing information, authentication information such as digital signature, authorization information such as account number, information for transaction management such as transaction id, etc. The SOAP body consists of the XML message to be transferred. It also contains error information if an error has occurred during the processing of a SOAP message.



**Figure 2.3 Structure of a SOAP message**

The SOAP encoding rules are applied only to SOAP RPC-style messages. They are not applied to SOAP document-style messages since these messages contain documents that are already in XML format. SOAP uses the XML Schema specification [11] to express the application-specific data types in XML. The application-specific data types can be either simple (i.e., scalar data types such as integer, string, etc.) or compound data types (such as arrays, etc.). To represent the scalar data types in XML, SOAP uses the built-in data types defined in XML Schema as well as new data types derived from the built-in data types. In order to represent the compound data types, such as arrays, in XML, SOAP uses its own convention.

The SOAP protocol can be used in combination with any transport protocol, such as HTTP, SMTP, FTP, etc. Currently, HTTP is the most commonly used transport protocol for transporting SOAP messages. Moreover, HTTP works very well with the

SOAP RPC messaging style since the SOAP RPC request message can be encapsulated within the HTTP request and the SOAP RPC response message within the HTTP response. SOAP makes use of XML namespaces [3, 6] to namespace-qualify the XML elements and attributes in the SOAP message and to refer to external schema definitions. SOAP tools that can be used to automatically generate SOAP messages as well as process them can be implemented in any programming language such as Java, Perl, Visual Basic, C, etc.

An example SOAP request message sent by a client to a service provider offering the BookPrice Web Service is shown in Figure 2.4 [9]. The request message consists of a SOAP envelope with a few namespace declarations and a SOAP body. Each namespace declaration consists of a namespace (i.e., a URI) and a namespace prefix that is assigned to the namespace. The namespace [“http://schemas.xmlsoap.org/soap/envelope/”](http://schemas.xmlsoap.org/soap/envelope/) consists of definitions of elements (such as “Envelope”, “Header”, “Body”, etc.) that are needed to create a SOAP message. The namespace declaration [“xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance”](http://www.w3.org/2001/XMLSchema-instance) indicates that this SOAP message is an XML document that conforms to an XML Schema. This namespace declaration also enables the use of elements defined in the namespace [“http://www.w3.org/2001/XMLSchema-instance”](http://www.w3.org/2001/XMLSchema-instance) within the SOAP message. The namespace declaration [“xmlns:xsd=http://www.w3.org/2001/XMLSchema”](http://www.w3.org/2001/XMLSchema) enables the use of elements and data types defined in the namespace [“http://www.w3.org/2001/XMLSchema”](http://www.w3.org/2001/XMLSchema) within the SOAP message. The SOAP body contains the method name “getBookPrice” and the method parameter i.e. the name of the book for which the client wants to know the price.

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:getBookPrice xmlns:ns1="urn:BookPriceService" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <bookName xsi:type="xsd:string">"Web Services"</bookName>
    </ns1:getBookPrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

**Figure 2.4 Example of a SOAP Request Message**

The XML-RPC packaging protocol [9] is another packaging protocol for Web Services. A detailed discussion of the XML-RPC packaging protocol is outside the scope of this thesis but it is mentioned here for completeness. It is a simpler protocol than SOAP and is mainly used for performing RPCs. Similarly to SOAP, it uses XML syntax to encode the request and the response message in a remote procedure call. XML-RPC uses HTTP as its transport protocol. For transferring the XML-RPC messages over a network between two diverse applications, the request messages are embedded within the HTTP request, while the response messages are embedded within the HTTP response. The XML-RPC request message consists of the remote method name and the method parameters. The response message consists of either the return values or error information in case of errors.

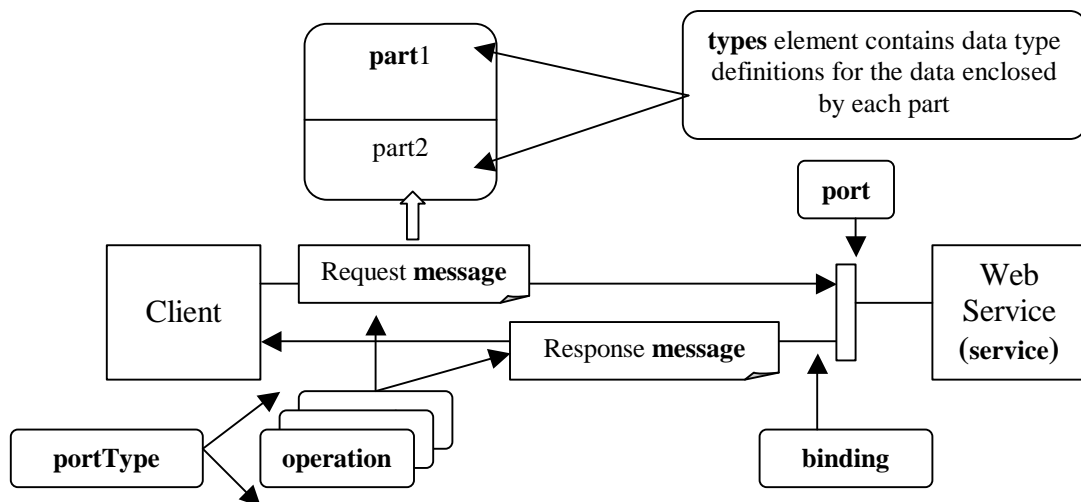
The next layer in the Web Services protocol stack, above the packaging layer is the description layer. This layer is associated with Web Service technologies such as Web Services Description Language (WSDL), Resource Description Framework (RDF), DARPA Agent Markup Language (DAML), etc. [26]. As mentioned in [26], both RDF and DAML have a richer syntax for describing Web Services than WSDL, but their

syntax is also very complex compared to that of WSDL. This makes the WSDL language more popular than RDF and DAML. The discussion of RDF and DAML is out of the scope of this thesis. WSDL is discussed in detail since it is very relevant to the thesis.

### 2.2.2 WSDL

WSDL [9, 10, 26] is an XML-based language for describing Web Services. The concept of a Web Service has been defined in Section 1.1. WSDL provides a standard mechanism for specifying the core functionality of a Web Service. WSDL also enables the specification of the location i.e. network address of a Web Service and methods for accessing the Web Service. Using the WSDL language, the Web Service providers can describe a common format of request their clients can use to access their Web Services.

Figure 2.5 [34], shows a client accessing a Web Service described using WSDL. The figure points out in bold the main elements of the WSDL language (discussed next) that are used to describe the Web Service.



**Figure 2.5 Client accessing a Web Service described using WSDL**



A single example, the buyStock Web Service, is used to explain the main elements of the WSDL language. The same example is also used in the rest of the thesis for other explanations. This example is discussed next followed by a discussion of the main elements of the WSDL language.

- **The buyStock Web Service Example**

The buyStock Web Service is a web service that enables its consumers to buy stocks of one or more companies. To buy stocks of a single company, a consumer sends a request message to the buyStock Web Service with the following parameters: stock symbol of the company, quantity of stocks to be bought, maximum price at which each stock should be bought, and date and time before which the stocks should be bought. On receiving the request from the consumer, the buyStock Web Service evaluates the stock market conditions i.e., whether the stock market is closed or open, etc. The Web Service buys stocks for the consumer only if the stock market conditions are appropriate. Whether the Web Service buys stocks or not, it sends a response message to its consumer. The response message consists of the following information: stock market closed or open, stock on sale or not, actual quantity of stocks bought, actual price at which each stock is bought, etc. Similar actions take place when a consumer wants to buy stocks of multiple companies.

The buyStock Web Service has been described in Appendix D using the WSDL language. The WSDL description consists of information such as operations supported by the buyStock Web Service (e.g., buySingleStockOperation), location of the buyStock Web Service, methods for accessing the buyStock Web Service, etc. For this WSDL

description of the buyStock Web Service, multiple service offerings offered by the buyStock Web Service, have been specified using the WSOL language. The WSOL description for the buyStock Web Service is given in Appendix E.

The main elements of the WSDL language are:

- **definitions** – The **definitions** element is a root element that contains definitions of all the other WSDL elements used to describe one or more Web Services. The **definitions** element also enables specification of various namespaces, including the ones that refer to external specifications such as WSDL, SOAP, XML Schema, etc.

The syntax for the **definitions** element is shown below:

```
<wsdl:definitions>
  < -- all other wsdl elements -- />
</wsdl:definitions>
```

For example, the **definitions** element defined for the buyStock Web Service is:

```
<wsdl:definitions name = "buyStockDefinition" xmlns:wsdl =
"http://schemas.xmlsoap.org/wsdl/" ...>
  <wsdl:import namespace = "... " location = "..."/>
  <wsdl:message name = "buySingleStockRequest">...</wsdl:message>
  <wsdl:portType name = "buyStockPortType">...</wsdl:portType>
  <wsdl:binding name = "buyStockBinding"...>...</wsdl:binding>
  <wsdl:service name = "buyStockService">...</wsdl:service>
  ...
</wsdl:definitions>
```

In the given example, the **definitions** element contains definitions of all the other WSDL elements that describe the buyStock Web Service.

- **types** – The **types** element provides the data type definitions for the data enclosed by the messages exchanged between a Web Service and its consumers. If the data enclosed within a message is of simple data types only such as string, integer, etc., then there is no need to specify the **types** element. But if a user-defined complex data

type needs to be defined, then this is done by using the **types** element. The **types** element provides the data type definitions using a type system. WSDL has the capability to support any type system. However, it is preferred to use XML Schema Definition (XSD) as the default type system for defining the data types in a message.

The syntax for the **types** element is shown below:

```
<wsdl:types>
  <xsd:schema/> <-- XSD type system -- >
</wsdl:types>
```

Other type systems can be used in the **types** element using extensibility elements as shown in the following syntax:

```
<wsdl:types>
  <-- type-system extensibility element -- />
</wsdl:types>
```

- **message** - The **message** element represents an abstract definition of the data to be transmitted between a Web Service and its consumer. The syntax for defining a message is as follows:

```
<wsdl:message name="...">
  <wsdl:part name="..." type="..." />
</wsdl:message>
```

A message consists of one or more parts. Each **part** element specifies the name of the part and the data type of that part. The data type could belong to any type system, such as, XSD. A part represents either an input parameter to an operation provided by the Web Service or its return value. For example, consider the specification of a **message** element for the buyStock Web Service:

```
<wsdl:message name = "buySingleStockRequest">
  <wsdl:part name = "symbol" type = "xsd:string"/>
  <wsdl:part name = "quantity" type = "xsd:nonNegativeInteger"/>
  <wsdl:part name = "maxPrice" type = "xsd:float"/>
  <wsdl:part name = "buyBeforeDateTime" type = "xsd:dateTime"/>
</wsdl:message>
```

In the given example, a message named “buySingleStockRequest” consists of four parts: “symbol” of data type “string”, “quantity” of data type “nonNegativeInteger”, “maxPrice” of data type “float”, and “buyBeforeDateTime” of data type “dateTime”.

- **portType** – The **portType** element corresponds to a set of one or more operations.

The syntax for defining a **portType** is as follows:

```
<wsdl:portType name="...">
  <wsdl:operation name="..."/>
</wsdl:portType>
```

Each **operation** in a **portType** refers to either an input message (with an **input** element) or an output message (with an **output** element) or a specific sequence of input/output messages (with **input/output** elements). WSDL supports four types of operations:

1. **One-way operation** – It refers to a single input message received by a Web Service. The syntax for this kind of operation is:

```
<wsdl:portType name="portType1">
  <wsdl:operation name="one-way-operation">
    <wsdl:input name="one-way-input" message="tns:message1"/>
  </wsdl:operation>
</wsdl:portType >
```

2. **Request-response operation** – It refers to an input message (request) received by a Web Service followed by an output message (response) sent by the same Web Service. This operation also supports an optional **fault** element that specifies an error message that may be output in case of errors. The syntax for this kind of operation is:

```
<wsdl:portType name="portType1">
  <wsdl:operation name="request-response-operation">
    <wsdl:input name="request-response-input"
      message="tns:message1"/>
    <wsdl:output name="request-response-output"
      message="tns:message2"/>
  </wsdl:operation>
</wsdl:portType>
```

```

        <wsdl:fault name="request-response-fault"
            message="tns:message3"/>
    </wsdl:operation>
</wsdl:portType >

```

For example, consider the specification of a **portType** element for the buyStock

Web Service:

```

<wsdl:portType name = "buyStockPortType">
    <wsdl:operation name = "buySingleStockOperation">
        <wsdl:input name = "input" message =
            "tns:buySingleStockRequest"/>
        <wsdl:output name = "output" message =
            "tns:buySingleStockResponse"/>
        <wsdl:fault name = "fault" message = "tns:buyStockFault"/>
    </wsdl:operation>
</wsdl:portType>

```

In the given example, a port type named “buyStockPortType” consists of a request-response operation named “buySingleStockOperation”. The operation refers to an input message “buySingleStockRequest”, an output message “buySingleStockResponse”, and a fault message “buyStockFault”. All these messages and their corresponding parts are defined prior to defining this port type.

3. **Solicit-response operation** – It refers to an output message (solicit) sent by a Web Service followed by an input message (response) received by the same Web Service. This operation also supports an optional **fault** element that specifies an error message that may be output in case of errors. The syntax for this kind of operation is:

```

<wsdl:portType name="portType1">
    <wsdl:operation name="solicit-response-operation">
        <wsdl:output name="solicit-response-output"
            message="tns:message1"/>
        <wsdl:input name="solicit-response-input"
            message="tns:message2"/>
        <wsdl:fault name="solicit-response-fault"
            message="tns:message3"/>
    </wsdl:operation>
</wsdl:portType >

```

4. **Notification operation** – It refers to a single output message sent by a Web Service. The syntax for this kind of operation is:

```
<wsdl:portType name="portType1">
  <wsdl:operation name="notification-operation">
    <wsdl:output name="notification-output"
      message="tns:message1"/>
    </wsdl:operation>
  </wsdl:portType >
```

- **binding** – A **binding** element specifies a message format, such as, SOAP, and a transport protocol, such as HTTP, or SMTP, that will be used for encoding and transmitting the messages to be exchanged between a Web Service and its clients. A binding is defined for a particular port type, i.e. for a particular set of operations defined in a port type. Several bindings with different transport protocols or different messaging protocols can be specified for a single port type. The syntax for defining a **binding** is as follows:

```
<wsdl:binding name="..." type="...">
  <!-- extensibility element -->
  <wsdl:operation name="...">
    <!-- extensibility element -->
    <wsdl:input name="...">
      <!-- extensibility element -->
    </wsdl:input>
    <wsdl:output name="...">
      <!-- extensibility element -->
    </wsdl:output>
    <wsdl:fault name="...">
      <!-- extensibility element -->
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
```

For example, consider the specification of a **binding** element for the buyStock Web Service:

```
<wsdl:binding name = "buyStockBinding" type = "tns:buyStockPortType">
  <soap:binding style = "rpc" transport =
    "http://schemas.xmlsoap.org/soap/http"/>
  ...
</wsdl:binding>
```

In the given example, a binding named “buyStockBinding” is specified for the port type “buyStockPortType”. This binding specifies that the messages (referred to by the operations specified within the “buyStockPortType”) to be exchanged between the buyStock Web Service and its clients will be encoded using the SOAP protocol and transmitted using the HTTP transport protocol.

WSDL supports message format specific bindings such as SOAP binding, HTTP GET/POST binding, and MIME binding. These bindings extend WSDL and hence the elements defined by these bindings are referred to as WSDL extensibility elements. Such extensibility elements have been shown in the syntax for **binding** element above. The discussion for HTTP GET/POST binding and MIME binding is out of scope of this thesis.

Since SOAP is a widely used protocol and supported by many Web Services, SOAP bindings are used more often to specify the SOAP message format specific details. For example, the `<soap:binding..../>` element is used to specify the style of the messages to be exchanged, whether RPC or document-oriented, and the transport protocol for transmitting those messages. The element `<soap:operation..../>` is used only if the transport protocol is HTTP because it can specify the value of the SOAPAction header in the HTTP request that is essential for determining the nature of the SOAP request. The element `<soap:body..../>` is used to specify how the parts of a specific message within a specific operation will be encoded within the body of the SOAP message for RPC style messages. For document-oriented messages no encoding of data is done, so no encoding style is specified.

- **port** - A **port** element specifies one or more network addresses where a particular Web Service is implemented and can be accessed from. It associates a network address with a particular binding i.e. with a particular set of message format and transport protocol. If a Web Service is implemented at multiple network addresses then a client can choose a particular network address for accessing the Web Service depending on a specific binding or a particular port type it wants to use. The syntax for defining a **port** is as follows:

```
<wsdl:port name="..." binding="...">
  <!-- extensibility element -->
</wsdl:port>
```

For example, consider the specification of a **port** element for the buyStock Web Service:

```
<wsdl:port name = "buyStockServicePort" binding = "tns:buyStockBinding">
  <soap:address location = "http://localhost:8080/soap/servlet/rpcrouter"/>
</wsdl:port>
```

In the given example, a port named “buyStockServicePort” is defined that specifies a single network address where the buyStock Web Service is implemented and can be accessed from. This port associates the network address with the binding “buyStockBinding” also defined for the buyStock Web Service.

A **port** element can enable specification of only one network address for a specific binding. A `<soap:address.../>` extensibility element is used within the **port** element to specify the implementation address of a Web Service if the Web Service supports the SOAP protocol.

- **service** – The **service** element groups a set of related ports together. It describes the name of a Web Service. It includes one or more port elements for the same Web



Service, each specifying a network address. Multiple Web services can be described in a single WSDL document. The syntax for defining a **service** is as shown below:

```
<wsdl:service name="...">
  <wsdl:port name="..." .../>
  <wsdl:port name="..." .../>
</wsdl:service>
```

For example, consider the specification of a **service** element for the buyStock Web Service:

```
<wsdl:service name = "buyStockService">
  <wsdl:port name = "buyStockServicePort" binding =
    "tns:buyStockBinding">...</wsdl:port>
</wsdl:service>
```

In the given example, a service named “buyStockService” is defined that consists of a single port named “buyStockServicePort” where the buyStock Web Service is implemented.

The WSDL language also defines some other elements besides the main elements. An example is the **import** element. The **import** element enables a WSDL document to import other WSDL documents or external XML Schema documents that define new data types. This increases the modularity and reusability of WSDL documents.

A number of tools are available that have support for WSDL, such as, IBM’s WSTK (Web Services Toolkit), Systinet’s WASP Server for Java, GLUE from “The Mind Electric”, Apache Axis, etc.

The discovery layer above the description layer in the Web Services protocol stack is associated with Web Service technologies such as Universal Description, Discovery, and Integration (UDDI), and Web Services Inspection Language (WS-

Inspection). UDDI [4, 9, 26] is a mechanism for publishing and discovering Web Services. UDDI consists of two main parts:

- 1) A specification for creating distributed Web Service registries and for publishing and discovering Web Services.
- 2) Implementations of the specification such as the UDDI Business Registry that enable the Web Service providers to publish the Web Services they offer and the Web Service consumers to discover Web Services.

A UDDI Business Registry contains information about a business and its Web Services, in three categories:

- White Pages that contain information about the name, contact address, phone number, description, etc. of a business.
- Yellow Pages that contain information about the classification of a business or its Web Services.
- Green Pages that contain information about the technical details of the Web Services such as pointers to WSDL descriptions of the Web Services, etc. Further details about UDDI are out of scope of this thesis.

The WS-Inspection language [26] enables a Web Service provider to advertise the Web Services it offers at a certain location on a network. The Web Service provider specifies the name of each Web Service and a pointer to its WSDL description in a WS-Inspection document in simple XML syntax. These WS-Inspection documents located on the Web Server of the service provider enable the Web Service consumers to discover the

Web Services and their locations. A detailed discussion of WS-Inspection language is out of scope of this thesis.

### **2.3 XML Parsing**

An XML document is parsed using a tool called an XML parser. An XML parser [31] is a software module that reads an XML document to be parsed, verifies that the content and structure of the XML document conforms to the rules specified in its corresponding grammar (specified using XML Schema [11] or DTD (Document Type Definition) [2]), and presents the structure and content of the XML document according to the Application Programming Interface (API) it uses. There are two types of APIs that are commonly used by XML parsers:

- DOM (Document Object Model) API

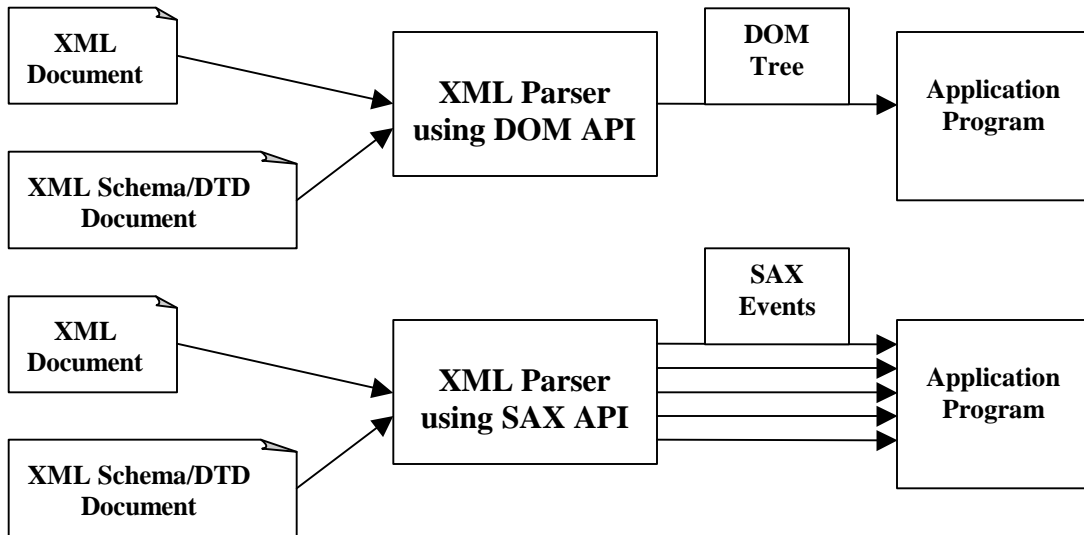
The DOM API enables the XML parser to present the structure and content of the XML document being parsed in the form of a tree. The DOM tree consists of several nodes that represent elements, text, etc. defined in the XML document. These nodes are arranged according to the structure of the XML document. The XML parser stores the DOM tree in memory. The DOM tree enables an application program to access and modify the XML document it represents.

- SAX (Simple API for XML) API

The SAX API enables the XML parser to present the structure and content of the XML document being parsed in the form of events. As the XML parser scans the XML document, it generates events such as `startElement`, `endElement`, etc. These events provide information such as element name, its value, its attributes, etc. as

defined in the XML document. The events are also structured according to the structure of the XML document being parsed. An application program handles these SAX events and processes them to extract the required information.

An overview of XML parsing is given in Figure 2.6.



**Figure 2.6 Overview of XML Parsing**

A number of XML parsers are available today from different vendors and in different languages such as XML Parser for Java [7], Microsoft XML Parser (MSXML), Xerces Java Parser [36], Xerces2 Java Parser [37], Xerces C++ Parser, Oracle's XML Parser for Java, etc. Since the Xerces2 Java Parser is used for implementing a prototype in the thesis, it is discussed in the next subsection. A discussion of all the other parsers is out of scope of this thesis.

## 2.4 Xerces2 Java Parser

The Xerces2 Java Parser [37] is an XML parser developed by the Apache Software Foundation (<http://www.apache.org>). It is a high performance parser fully compliant with Java. It belongs to Xerces2 (the latest version of Xerces in the Apache Xerces family, where Xerces is a sub-project of the Apache XML project and is focused on the development of XML parsers in Java, C++, etc.). Xerces2 defines a framework called Xerces Native Interface (XNI) that enables the construction of new parser components, parser configurations, and parsers. The Xerces2 Java parser implementation is written using the XNI framework.

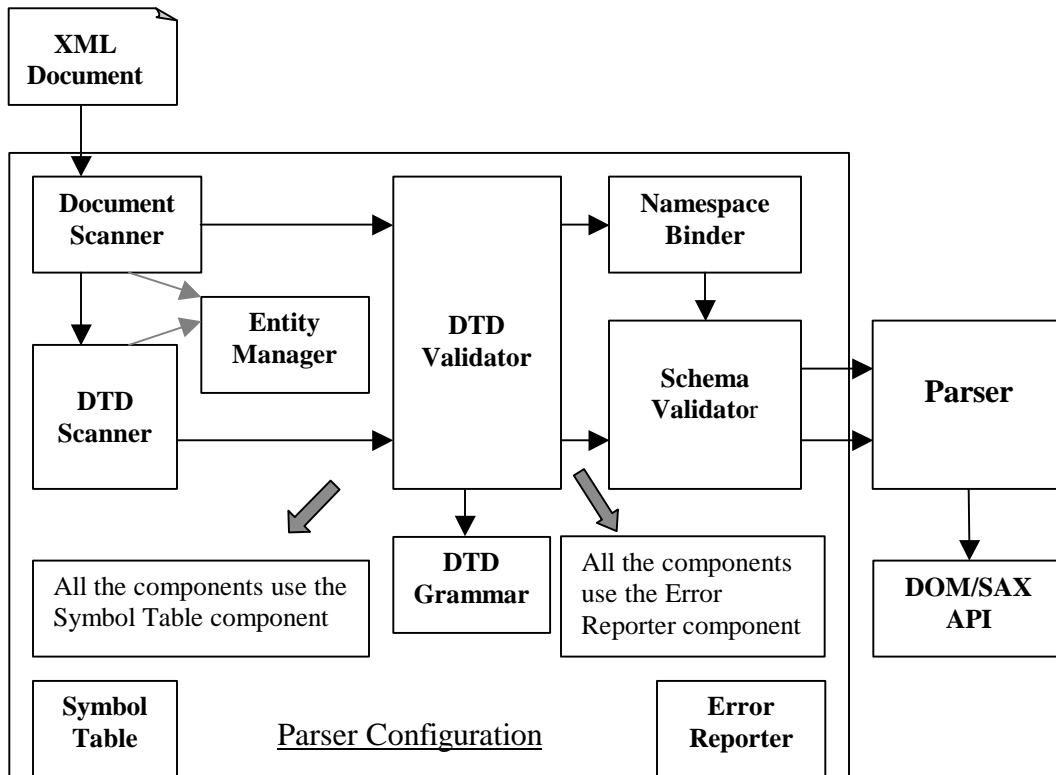
I have taken a look at several XML parsers, such as, XML4J, Xerces Java Parser, and Xerces2 Java parser. Among all these parsers, I have chosen the Xerces2 Java Parser, made available by the Apache XML Project, to implement the prototype parser in my thesis. There are several reasons behind this choice.

First of all, the Xerces2 Java Parser is an open source parser. So, its source code could be studied. The working of the parser could also be understood. This could eventually help in designing a prototype parser for my thesis. Secondly, the Xerces2 Java Parser is a fully conforming XML Schema processor. The prototype parser that is to be created in my thesis has to parse documents against a grammar developed using XML Schema. Since, the Xerces2 Java Parser can fully support validation against XML Schema, the source code implementing this feature could be re-used in the prototype parser. Also, since the Xerces2 Java Parser has a very clean and modular code, it is easy to understand the implementation of the parser. Moreover, I have chosen Xerces2 Java Parser because it supports standards and APIs such as XML Schema, XML Namespaces,

DOM, SAX, etc. Also, because it is written in Java, it could be re-used to write the prototype parser in my thesis that will also be written in Java.

Other XML Parsers that I have studied do not have all the features supported by the Xerces2 Java Parser. So, I have chosen to implement my prototype parser based on the Xerces2 Java Parser.

The design details of the Xerces2 Java parser are shown in Figure 2.7 [37].



**Figure 2.7 Design details of Xerces2 Java Parser**

The Xerces2 Java parser uses a parser configuration to parse an XML document. This parser configuration implements one of the parser configuration interfaces (such as XMLParserConfiguration) provided by the XNI framework. The parser configuration used by Xerces2 Java parser comprises of several parser components such as:

- Document Scanner

The Document Scanner component scans the XML document and passes the document information to the DTD Validator component.

- DTD Scanner

The DTD Scanner component scans the XML document and passes the DTD information to the DTD Validator component. The XML document contains DTD information (such as name and location of the DTD grammar) only if the XML document conforms to a DTD grammar. If the XML document conforms to an XML Schema, then the XML document contains information about the XML Schema.

- DTD Validator

If the XML document conforms to a DTD grammar, then the DTD Validator component receives both the document information and the DTD information. The DTD Validator component loads the DTD grammar specified in the DTD information and validates the structure and content of the XML document against the DTD grammar.

- Namespace Binder

The Namespace Binder component detects the namespace bindings in the elements and attributes defined in the XML document and emits the start and the end prefix mapping events.

- Schema Validator

If the XML document conforms to an XML Schema, then the Schema Validator component loads the XML Schema (the name and location of the XML Schema is

specified in the XML document) and validates the structure and content of the XML document against the XML Schema.

- Entity Manager

The Entity Manager component enables the Document Scanner and the DTD Scanner to operate smoothly even when entities go in and out of scope. It handles the starting and stopping of entities automatically.

- Symbol Table

The Symbol Table component stores symbols (i.e. common strings appearing in the document) and returns a reference to each string stored. This reference is used by other parser components to track the strings.

- Error Reporter

The Error Reporter component enables all the other parser components to report errors to the Error Handler registered with the parser configuration.

These parser components are managed by a component manager. The component manager keeps track of settings of the parser such as selection of schema validation feature, etc. It instantiates and configures the parser components. It also assembles various parser components in a pipeline for parsing, and initiates parsing of documents.

The parser configuration as shown in Figure 2.7, consists of two separate pipelines of components: one is for the flow of document information and the other is for the flow of DTD information. The document information flows from the Document Scanner component to the DTD Validator component. From there it flows to the Namespace Binder component and reaches the Schema Validator component that is the last component in the pipeline. The document information handled by the document



handler registered with the parser configuration is then forwarded by the parser configuration to the parser which represents the information in the form of a DOM tree or SAX events depending on the requirements of the application using the parser. The DTD information flows along the same path as that of document information except that it bypasses the Namespace Binder component. The XNI framework enables the information to flow from one component to another by setting the input document handler and the output document handler for each component in the pipeline. The separation of the parser from the parser configuration enables the same parser to be used with different parser configurations comprised of different parser components.

## **2.5 Summary**

A basic Web Services Architecture and its main components were discussed in this chapter. Several Web Service technologies were reviewed including SOAP and WSDL in detail and XML-RPC, UDDI, and WS-Inspection language in brief. An overview of XML parsing was provided. Design details of the Xerces2 Java parser were given and the reasons for choosing it to implement the prototype parser in this thesis were also stated.

### 3 Web Service Offerings Language (WSOL) and its Related Works

---

WSOL (Web Service Offerings Language) is a novel language for specification of constraints and classes of service for Web Services. Vladimir Tasic [40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50], a Ph.D. candidate in our research group at the Network Management Lab (Carleton University), is developing the architecture of the WSOL language. He decided what concepts and features would be built into WSOL and why. My Masters thesis contributes to the development of the XML grammar for WSOL and the corresponding WSOL parser. The development of WSOL concepts and my thesis work was parallel and cyclic. This parallelism and frequent changes created difficulties, but they also enabled that feedback from my work could influence the architecture of WSOL. Since the WSOL language is central to this thesis, this chapter discusses it in detail.

When development of the WSOL started, there was no language to comprehensively specify constraints and classes of service for Web Services. Languages and works related to WSOL started being developed in parallel to WSOL. These languages are still under development. The most important such languages and works are the Web Service Level Agreement language (WSLA), Web Service Management language (WSML), DARPA Agent Markup Language for Web Services (DAML-S), Web Services End-Point Language (WSEL), XL - an XML programming language for Web Service specification and composition, CPP and CPA concepts of ebXML (Electronic Business using XML), etc. A brief description of the main concepts of these languages, their tools, and ways in which they differ from WSOL is given in this chapter.

At the end of this chapter, a comparison between WSOL and the works most relevant to WSOL has been summarized.

### **3.1 WSOL**

WSOL [44, 45, 50] is an XML-based language that enables formal specification of one or more service offerings for a Web Service. A service offering represents a single class of service for a Web Service. A class of service is defined by a combination of various constraints and statements. WSOL supports specification of constraints, such as functional constraints (pre-conditions, post-conditions, and future-conditions), non-functional constraints (a.k.a. QoS constraints), and access rights. WSOL supports specification of statements, such as price, price default, penalty, penalty default, subscription, external operation call, management responsibility, Include, and Instantiate.

The WSOL language is fully compatible with the WSDL language described in Section 2.2.2. There is no overlap between the two languages. They can be used together without special translation. The WSOL language enables specification of multiple service offerings i.e. multiple classes of service for one WSDL description of a Web Service.

The WSOL language defines the following constructs to enable formal specification of multiple service offerings for a Web Service:

#### **3.1.1 Constraint**

A constraint could be a functional constraint, a non-functional constraint, or an access right. Each constraint is in the form of a Boolean expression, specifying some condition

that is to be verified before and/or after the execution of the Web Service operation for which the constraint is specified. Different types of constraints are discussed below:

◆ **Functional constraint**

A functional constraint refers to some condition that needs to be satisfied for the execution of a Web Service operation to be functionally correct. Functional constraints can be evaluated by third parties. WSOL currently supports specification of several functional constraints such as pre-conditions, post-conditions, and future-conditions [45]. A pre-condition is some condition that is specified on one or more input message parts of the invoked operation of the Web Service. All pre-conditions defined for an invoked operation are evaluated before executing that operation to check whether the consumer has specified the input parameters correctly or not. A post-condition is some condition that refers to one or more output message parts of the invoked operation of the Web Service. It can, but need not, also refer to one or more input message parts. All post-conditions defined for an invoked operation are evaluated immediately after executing that operation to check whether the operation executed correctly or not. A future-condition is similar to a post-condition except that it is evaluated on a specific date or time or after a specific duration of time after the execution of the invoked operation is over.

◆ **Non-functional constraint**

A non-functional constraint is some condition that refers to one or more non-functional properties of an operation invocation such as performance, reliability, availability, etc. These non-functional properties do not define functional

correctness but can be important for the consumer. Non-functional constraints are also known as QoS constraints. A WSOL “QoS metric” defines one or more non-functional characteristics of an operation invocation, such as ResponseTime, PerformanceAndAvailability, etc. Some QoS metrics in WSOL can be aggregations of other QoS metrics. All the WSOL QoS metric types will be defined in external ontologies in future. For evaluating whether a specified QoS constraint is satisfied or not, the QoS metrics specified within the QoS constraint are monitored by some management entity. This management entity could be a Web Service supplier, consumer, or a trusted third-party.

◆ **Access Right**

An access right refers to a condition under which a consumer using a particular service offering has the right to access a particular operation. Access is allowed only if an access right constraint for the operation is explicitly specified and satisfied. If no access right for an operation is specified inside a service offering, this implies that access is forbidden. Access rights limit access to operations and ports of a Web Service and are used in WSOL for service differentiation. They are not used for addressing security issues but they can be a small part of a security management solution for Web Services.

### **3.1.2 Statement**

A statement construct enables the specification of important information such as price, penalty, management responsibility, etc. A statement in WSOL could be a price/penalty statement, a price/penalty default statement, a subscription payment model statement, an

external operation call statement, a management responsibility statement, an Include statement, or an Instantiate statement. They are discussed below:

◆ **Subscription statement**

A subscription statement is specified for a subscription-based service offering. It states the price that has to be paid by a Web Service consumer for using a particular service offering for a specified duration of time.

◆ **Price statement**

A price statement refers to the amount of money that a Web Service consumer has to pay to a supplier Web Service for using a particular operation of the Web Service. The price is paid only if the Web Service is successful in fulfilling all the constraints specified in the service offering being used by the consumer. The price statement supports the subscription payment model, the pay-per-use payment model, and their combinations.

◆ **Price default statement**

A price default statement refers to the default amount of money that a Web Service consumer has to pay to a supplier Web Service for any one operation invocation of a service offering. The default price is paid only if the Web Service is successful in fulfilling all the constraints specified in the service offering being used by the consumer. The price default statement differs from the price statement in that it is not specified for a particular Web Service operation. When it is specified, it refers to any Web Service operation. A price statement for a particular operation overwrites a price default statement.

◆ **Penalty statement**

A penalty statement refers to the amount of money that a supplier Web Service has to pay to a Web Service consumer if it could not fulfill the constraints, such as, postconditions, future conditions, and QoS constraints, specified for a particular operation being used by the consumer. The penalty statement supports the subscription payment model, the pay-per-use payment model, and their combinations.

◆ **Penalty default statement**

A penalty default statement refers to the default amount of money that a supplier Web Service has to pay to a Web Service consumer if it could not fulfill the constraints, such as, postconditions, future conditions, and QoS constraints, specified for any one operation invocation of a service offering. The penalty default statement differs from the penalty statement in that it is not specified for a particular Web Service operation. When it is specified, it refers to any Web Service operation. A penalty statement for a particular operation overwrites a penalty default statement.

◆ **External Operation Call statement**

It enables the specification of an external operation call i.e. call made by the constraint-checking code during evaluation of constraints to an operation supported by some other Web Service. The statement describes the called operation and also specifies its implementation address. It also specifies values of input parameters to the external operation.

◆ **Management Responsibility statement**

It specifies the management responsibility of a management entity. A management entity could be the Web Service supplier, the Web Service consumer, or an independent third party trusted by the supplier and the consumer. The third party acts as an intermediary that intercepts the SOAP request and response messages between the supplier and the consumer. The management responsibilities of the management entities include tasks such as checking a particular constraint, a constraint group (CG) inside a service offering, or a complete service offering.

◆ **Include statement**

It enables constraints, statements, or constraint groups (CGs) to be reused across different service offerings, constraint groups, and/or constraint group templates.

◆ **Instantiate statement**

It enables a constraint group template (CGT) to be instantiated within a constraint group (CG), another CGT, a service offering, or outside service offerings.

### **3.1.3 Constraint Group (CG), Constraint Group Template (CGT), Service Offering**

- **constraint group (CG)**

A constraint group (CG) is a named construct that enables grouping of constraints, statements, nested CGs, and instantiations of CGTs (The definition of a CGT is given after the explanation of the constraint group construct). A CG can contain new



definitions of constraints, statements, and/or CGs. A CG can also contain, using the “include” mechanism, those constraints, statements, and/or CGs which are defined elsewhere. A new CG can also be defined as an extension of some existing CG, using single inheritance of CGs. A derived CG contains all constraints, statements, and CGs from the base CG and additional ones. The constraint group construct provides several benefits such as:

1. A CG can be re-used across several service offerings.
2. All the constraints, statements, and/or CGs within a CG can be managed by a single management entity.
3. Names of constraints, statements, and/or CGs within a CG can be re-used within other CGs.

- **constraint group template (CGT)**

The CGT (constraint group template) construct in WSOL is defined as a parameterized CG. It is similar to a CG except that it contains one or more abstract parameters defined before the definition of constraints, statements, and/or CGs. A CGT parameter has a name, a data type, and a measurement unit (specified only if the data type is “numberWithUnit”). The data type “numberWithUnit” consists of a numeric constant followed by a measurement unit such as “millisecond”. A parameter defined within a CGT can be a part of the Boolean expressions contained within the constraints defined in the same CGT. Instantiating a CGT, by providing constant values for all its parameters, results into a new CG. Multiple instantiations of a CGT with different parameter values results in multiple different CGs. The WSOL CGT

construct has several limitations. A new CGT cannot be defined within another CGT. The constraints defined within a CGT and having the CGT parameters as part of their Boolean expressions cannot be included within other CGs, CGTs, or service offerings.

- **service offering**

The service offering construct in WSOL enables grouping of constraints, statements, CGs, and instantiations of CGTs defined for the same Web Service. A service offering in WSOL is also referred to as a component-level service offering (CLSO). A service offering is almost syntactically equivalent to a big CG. Therefore, the rules that apply to a CG also apply to a service offering except that CGs can be nested but service offerings cannot. A new service offering can be defined from an existing one using the single inheritance (extension) mechanism supported by WSOL. Multiple service offerings of a Web Service can be defined and offered to Web Service consumers. The consumers can select and utilize only one service offering at a time.

WSOL also supports specification of relationships between service offerings that are dynamic in nature (i.e., they can change during run-time). The relationship between a service offering (SO1) and its replacement service offering (SO2) can be specified in the form of <SO1, S, SO2> which means that if a set of constraints and/or CGs (represented by “S”) defined in service offering “SO1” are not satisfied, then the service offering “SO1” can be replaced by the service offering “SO2”. An example of such specification is given in Section 4.2. These relationships are specified in a special XML format outside WSOL files so that they can be independent of all other WSOL constructs. The

relationships are also specified outside WSOL files so that they can be updated easily with new information (such as when a new service offering is being created dynamically). As mentioned in [44, 45, 50], this concept of specifying dynamic relationships between service offerings has been introduced in WSOL to enable two purposes:

A) Dynamic adaptation of Web Service compositions to various changes.

The dynamic adaptation can be achieved by manipulating the service offerings i.e. by switching between service offerings, deactivating/reactivating existing service offerings, and creating new appropriate service offerings.

B) Negotiation and selection of a replacement service offering.

### **3.1.4 Benefits of WSOL**

The main benefits of WSOL are as follows:

- Expressive power
  - WSOL enables formal specification of different categories of constraints and management statements, as well as multiple classes of service for one Web Service.
  - WSOL enables reuse of specifications with the help of several built-in mechanisms:
    1. CGTs
    2. Single inheritance of service offerings, CGs, and CGTs
    3. The *<include>* statement

4. Grouping of constraints, statements, and nested CGs into CGs and CGTs
- WSOL supports specification of both static and dynamic relationships between service offerings.
  - WSOL is extensible because it enables specification of new types of constraints with the generic *<constraint>* element.
- Reduced run-time overhead
    - WSOL enables specification of a class of service (represented by a service offering), instead of more demanding alternatives such as custom-made SLAs, user profiles, and others.
    - WSOL enables specification of various categories of constraints (i.e., functional constraints, QoS constraints, and access rights) and management statements using one single language which results in a lower overhead than the overhead incurred when different languages are used for specification of various categories of constraints and management statements.
    - Metering of QoS metrics and evaluation of WSOL constraints when delegated to specialized third parties (i.e., SOAP intermediaries or probes) reduces the run-time overhead at the supplier Web Service and its consumers.
    - Reasoning about WSOL service offerings, for example in the process of selection and negotiation of service offerings, when delegated to specialized third parties can also reduce run-time overhead.
  - Orientation towards management applications
    - WSOL enables formal and unambiguous specification of constraints in a format that can be useful for automatic generation of constraint-checking code.

- WSOL enables specification of management statements within service offerings such as:
  1. management responsibility statements, and
  2. statements about prices and monetary penalties.
- WSOL supports specification of management third parties in:
  1. management responsibility statements, and
  2. the *measuredBy* attribute of QoS metrics.
- WSOL supports specification of accounting parties.
- WSOL supports specification of dynamic relationships between service offerings.

### **3.2 Related Works**

WSOL is not the only language that enables specification of constraints and contracts for Web Services. Languages and works such as WSLA, WSML, DAML-S, WSEL, XL, etc. also enable such specification. These languages and works related to WSOL are being developed concurrently to WSOL. Moreover, none of these WSOL related languages and works is an alternative to WSOL. They only address partially the issues similar to issues addressed by WSOL. Each of these languages and works related to WSOL is discussed and compared with WSOL in the following subsections. First, the works most relevant to WSOL are discussed. These works include WSLA, WSML, DAML-S, XL, and WSEL. Then, the works less relevant to WSOL are discussed such as WSOL related parts of ebXML, and other works. Finally, the works least relevant to WSOL such as QML and DBC are discussed.

### **3.2.1 Works Most Relevant to WSOL**

#### **3.2.1.1 WSLA**

The WSLA language, developed by IBM [14] is an XML-based language designed to enable formal specification of service level agreements for Web Services. A service level agreement (SLA) in WSLA consists of three types of information:

- a) Information regarding the parties involved in the agreement, their roles i.e. service provider, consumer, or third parties, and their individual management actions exposed to the other parties participating in the contract.
- b) Information regarding the service, such as service level parameters (e.g., response time, throughput, etc.) commonly understood by all the parties involved and for which the service level guarantees are provided.
- c) Information regarding the obligations i.e. service and action guarantees of each party involved in the agreement.

WSLA defines two main types of parties:

- ◆ Signatory parties such as service provider and service consumer. They are the main parties in a service level agreement.
- ◆ Supporting parties such as third parties which are sponsored by one or both of the signatory parties. The supporting parties provide services such as measurement, condition evaluation, management, supervision of service guarantees provided by the service provider, etc.

Information about each party in a SLA includes its contact information, its name, and a set of actions. An example of an action is notification performed when particular events, such as when the service guarantees are not satisfied, occur.

WSLA enables the definition of one or more service level agreement (SLA) parameters. An SLA parameter describes a property of a service that can be observed, such as response time, throughput, etc. The value of this observable property of service is obtained from a source of measurement. A “Metric” in WSLA, describes exactly what an SLA parameter means by specifying how to compute the value of an SLA parameter. WSLA enables specification of how the SLA parameters are aggregated from basic metrics, who is responsible for measuring the basic metrics, and how the SLA parameter values are obtained. WSLA also enables the specification of operations implemented by the service provider. The descriptions of these operations contain the observable properties of the operations and binding-specific information for accessing the operations.

WSLA enables the description of obligations of each party participating in a service level agreement. There can be two types of obligations: Service level guarantees are guarantees that the SLA parameters will be in a particular state within a given time period. For instance, guarantee that the SLA parameter “response time” of a particular operation will be less than 10 milliseconds between 10 AM and 5 PM. The service level guarantees are provided mainly by the service provider. The other type of obligation is an Action guarantee - a guarantee to take some action in a particular situation. For example, guarantee to send some notification in the case of violation of a service level guarantee. The action guarantees can be provided by any party including the supporting parties.

Several service level guarantees and action guarantees can also be grouped together for creating templates of groups of guarantees, for associating monetary penalties with groups, etc. WSLA also enables optional specification of cost of services as well as associated penalties.

The grammar for the WSLA language has been defined in XML using the XML Schema language. It enables creation of WSLA documents conforming to the grammar. A tool called “SLA Compliance Monitor” has been developed for the WSLA language by IBM [25]. This tool is included in the IBM Web Services Toolkit (version 3.2). The WSLA language, like the WSOL language, enables the specification of some QoS constraints, management information, and price/penalty for Web Services. But it is different from WSOL in several ways. WSLA does not have the concept of multiple service offerings for a Web Service. So, it is unable to provide the service providers and consumers with all the benefits resulting from multiple service offerings. It does not enable providing information about any functional constraints, access rights, external operation calls, etc.

#### **3.2.1.2 WSML**

The WSML language, developed by HP [33] is an XML-based language that enables formal, precise, unambiguous, and flexible specification of service level agreements (SLAs) for Web Services. They view an SLA as a contract between two Web Services that consists of certain guarantees offered by one Web Service (supplier) to the other (consumer).



As stated in [33], precise and flexible specification of SLAs is essential for automated SLA management (i.e., for automatic monitoring, enforcement, and optimization of SLAs between Web Services) and for making the specification of SLAs extensible so that new unforeseen SLA specifications can be specified from existing ones. WSML is fully compatible with WSDL and WSFL (Web Services Flow Language).

An SLA specified in WSML consists of the following information:

- A date constraint,
  - Consists of a start date, an end date, and a next evaluation date for the SLA.
- Parties involved in the SLA, and
- A set of SLOs (Service Level Objectives).
  - Each SLO consists of a day-time constraint and a set of clauses.
    - A day-time constraint specifies the days and time during the week when the SLO will be in effect (e.g., Mo-We, 6:00PM-8:00PM).
    - Each clause in an SLO specifies a measured item, the time or event when the clause will be evaluated, the set of samples of the measured item (e.g., the 5 longest running transactions) on which the clause will be evaluated, a mathematical function (e.g., response time function) and a condition (e.g., response time function < 10 ms) that will be applied on the set of samples, and an action that will be activated.
      - A measured item specified within a clause states the names of one or more monitored items (e.g., a message, a set of operations, a port, etc.) across which the same measurement will be applied. For each monitored item, information such as the location (i.e., service provider side or

consumer side) where the item will be measured, the construct type of the item, and the construct reference is also given.

The grammar for the WSML language has been defined in XML using the XML Schema language. It enables creation of WSML documents conforming to the grammar. Several tools have been developed for the WSML language by HP [32]. WSML is also used in the HP OpenView Web Services Management Engine (WSME). The WSML language, like the WSOL language, enables specification of QoS constraints and some management information for Web Services and is fully compatible with WSDL. But unlike WSOL, WSML does not enable specification of multiple service offerings for a Web Service. It also does not enable specification of functional constraints and access right that can be specified in WSOL.

### **3.2.1.3 DAML-S**

DAML-S [38, 39] is a language that enables semantic description of Web Services. DAML-S defines an ontology for Web Services that enables the specification of various semantic aspects of Web Services such as its properties, capabilities, etc. This ontology is defined using DAML syntax. The goal of DAML-S is to enable complex tasks such as automatic discovery of Web Services, automatic invocation of Web Services, automatic composition and interoperation of Web Services, automatic monitoring of Web Services, verification of properties of Web Services, etc.

The DAML-S ontology defines several classes and properties to enable semantic description of Web Services. The class “Service” is defined at the top of the ontology. It has three properties: “presents”, “describedBy”, and “supports”. The class “Service” is

related through its properties to three other classes defined in the DAML-S ontology. It is related to class “ServiceProfile” through its “presents” property, to class “ServiceModel” through its “describedBy” property, and to class “ServiceGrounding” through its “supports” property. This means that in DAML-S, a service presents a service profile, a service is described by a service model, and a service supports a service grounding. The classes “ServiceProfile”, “ServiceModel”, and “ServiceGrounding” are discussed in detail below:

- **ServiceProfile**

The class “ServiceProfile” enables the description of the profile of a Web Service. The profile consists of three types of information: i) information about the Web Service including name of the Web Service, service provided by the Web Service, a link to the actors such as provider, requestor, etc. involved in the execution of the Web Service, provider and requestor of the Web Service, etc. ii) information regarding the functional behaviour of the Web Service, and iii) additional information about the Web Service.

Additional information about a Web Service includes information such as:

1. Geographic Scope – geographic area in which the Web Service can provide its service.
2. Degree of Quality – the degree of quality provided by the Web Service e.g. fastest service, cheapest service, etc.
3. Web Service Parameters – parameters such as invocation cost, average response time, etc.

4. Communication Language – language the Web Service uses to communicate e.g. SOAP.
5. Type of Service - the type of Web Service such as B2B (Business-to-Business), B2C (Business-to-Consumer), etc.
6. Category of the Web Service – the category the Web Service belongs to e.g. Delivery Service, Information service, etc.
7. Quality of Service Guarantees – guarantees that the Web Service promises to deliver, such as guarantee to provide the service with a response time of less than 10 seconds.
8. Quality Rating – ratings that refer to industry-based ratings of the Web Service.

All the above mentioned additional information is specified with help of some functional attributes. The Web Service providers use the “ServiceProfile” class to advertise the Web Service offered by them. The Web Service consumers also use the “ServiceProfile” class to specify the type of service they are looking for. The consumers looking for a specific Web Service send their requests to a service registry that then tries to match them with the profiles of several service providers and notify the consumers if a match is found.

- **ServiceModel**

The class “ServiceModel” enables the description of how a Web Service works. In DAML-S, a service can be viewed as a process. So, the DAML-S ontology defines a subclass of the “ServiceModel” class called “ProcessModel” that enables the

description of the workflow of a Web Service using its subclasses and their properties. The subclasses of the “ProcessModel” class are grouped into two ontologies: Process Ontology and Process Control Ontology. The Process Ontology consists of a class called “Process” at the top of its ontology. Using this class and its properties, a Web Service can be described in terms of its inputs, outputs, participants, preconditions, and side effects (that result from execution of the Web Service). Using the “CompositeProcess” class in the Process Ontology, a Web Service can be described in terms of the subprocesses it is composed of, if any. The order and condition of execution of these subprocesses such as sequence, parallel, random order, etc. can also be specified using the properties and subclasses of the “CompositeProcess” class. The Process Control Ontology enables the description of the state of the Web Service. The Web Service could be in an initial activation state, execution state, or in a completion state.

- **ServiceGrounding**

The “ServiceGrounding” class enables the description of how a Web Service can be accessed. It enables the specification of the transport protocol, the message format, etc. to be used to invoke the Web Service. DAML-S, when used with the WSDL language, can ground a Web Service. This is possible by i) extending the WSDL description of a Web Service with some DAML-S specific attributes, and ii) including within the Web Service’s DAML-S specification, a reference to specific WSDL elements defined in the corresponding WSDL document.

The DAML-S language currently does not have any tool developed specifically for it such as a DAML-S parser. Since the DAML-S specification is expressed using the DAML syntax, any tool developed for DAML (e.g., the DAML Validator) can be used for DAML-S as well. More details about the DAML tools are given on-line at [“http://www.daml.org/tools/”](http://www.daml.org/tools/).

The DAML-S language is a work in progress. Like the WSOL language, it enables specification of pre-conditions, post-conditions, some QoS constraints, and price of Web Services. But it is different from WSOL in several aspects. DAML-S does not enable specification of multiple service offerings for a Web Service. So it is unable to provide the service providers and consumers with all the benefits resulting from multiple service offerings. It does not enable providing any information about access rights, management responsibilities, etc. It also does not specify different constructs such as constraint, constraint group, etc. as specified in WSOL.

#### **3.2.1.4 WSEL**

WSEL (Web Service Endpoint Language) is a language being developed by IBM. WSEL has been “work in progress” for quite a long time. The WSEL language [15] has the goal of specifying the behavior supported by a Web Service. The behavior supported by a Web Service could be the QoS being offered by the Web Service, information regarding the proper usage of the interface supported by the Web Service, etc. The intention is to build this behavioral specification for a Web Service on top of the interface specification specified in the WSDL language. Since the WSEL language is still under development,

no tool has been developed for it and no further details about the language have been published so far.

### **3.2.2 Works Less Relevant to WSOL**

#### **3.2.2.1 CPP and CPA concepts of ebXML**

The ebXML specification [20, 21] defines the rules for doing electronic business. It consists of several different concepts, but, only those concepts that are related to WSOL are discussed in this subsection. ebXML defines two concepts: Collaboration Protocol Profile (CPP) and Collaboration Protocol Agreement (CPA). CPP enables any business organization to describe its profile i.e. to describe the business processes (Web Services) it supports, the messages exchanged, the transport protocol for the messages, the role it plays in a business process, etc. The business organization publishes its profile with the ebXML Repository and enables other business organizations to access the profile. The concept of CPP is similar to WSDL with which the WSOL language is compatible. CPP enables specification of parameters similar to parameters specified by WSDL (e.g. Web Service name, its network address, etc.), and also additional parameters. CPA is a business contract between two business organizations that wish to conduct business. The contract consists of all of the agreed-upon terms by both the organizations such as messaging protocol to be used, transport protocol to be used, etc. CPA is also used to verify that the business transactions between the two organizations are according to the business contract.

ebXML, like the WSOL language, enables the specification of contracts for Web Services. But it does not have the concept of multiple service offerings for a Web Service. So it is unable to provide the Web service providers and consumers with all the benefits resulting from multiple service offerings. Also, the CPA in ebXML does not provide information about QoS constraints, price/penalty, etc. as provided by WSOL.

### **3.2.2.2 XL**

XL [16] is a high-level and declarative XML programming language mainly designed for specification and composition of Web Services. It is being developed at Technical University, Munich. XL is advertised as a simple, platform independent language with easily understood syntax and semantics. XL is currently fully compliant with several W3C standards such as XML Schema, XML Query (XQuery), and XML Protocol. The main goals of the XL language are:

- To support a unique XML-based data model and type system: XQuery.
- To enable message-based programming. XL has the intention to enable Web Services written in XL to communicate with each other as well as with Web Services written in other programming languages via SOAP messages.
- To support conversations between two or more Web Services.
- To enable basic SOAP Web Services written in any programming language (e.g., Java, XL, etc.) to be composed into high-level Web Services.
- To enable the specification of actions such as logging, error handling, workload management, etc. that are particularly carried out during the implementation of a Web Service.



- To enable specification of transactions (i.e., sequences of actions to be executed).
- To enable specification of authentication, authorization, and security information in order to control access to a Web Service.

The XL programming language has been developed by extending the XQuery language. The XQuery language enables specification of XML expressions and XML Queries. XL enables the specification of logic of complex Web Services that could not be described with the simple XQuery language. Specification of a Web Service in XL includes specification of local function definitions, local data declarations, declarative Web Service clauses, and Web Service operations. The local data declarations can contain local type definitions, imported schema components, and local variables. The declarative Web Service clauses can be “history”, “defaultoperation”, “unknownoperation”, “init-close”, “invariants”, “on change”, “on event”, “on error invoke”, “conversation pattern”, and “conversationtimeout”. These clauses can be defined for controlling the global behavior of Web Services, interaction between two or more Web Services, and execution of Web Service operations.

A Web Service operation, when specified in XL, is composed of the declarative operation clauses and the operation body. The declarative operation clauses control the run-time behavior of the operations. Some of the clauses are similar syntactically and semantically to the Web Service declarative clauses mentioned earlier such as “history”, “conversation pattern”, etc. Some clauses are specific only to operations, such as “preconditions” (that define conditions that are checked before the execution of an operation), “postconditions” (that define conditions that are checked after the execution

of an operation), and “no sideeffects” (that define that the operation does not have any sideeffects on the internal state of the Web Service).

The operation body can consist of one or more statements. Statements represent an extended version of XQuery expressions. XL provides different types of statements, such as variable assignment statements, assertion statements, conditional statements, iteration statements, loops, error handling statements, return statements. XL also provides XML specific statements, such as update statements to manipulate XML data. It also provides Web Service specific statements, such as Web Services invocation statements (that can be used for invoking or sending messages to another SOAP Web Service written in any programming language such as XL, Java, etc.), logging statement, halt statement, sleep statement, wait on event and wait on change statements, retry statements, etc. XL also provides different types of statement combinators. These combinators can combine two or more statements together such as sequence, dataflow, parallel execution, choice, etc.

The XL language extends the simple XQuery expression language. The XL language is still under development. There are many design goals of XL that have not been accomplished yet. A prototype XL compiler in C++ is also under development. More information regarding the prototype XL compiler is given on-line at “[http://mozart-dev.sourceforge.net/xl\\_status.html](http://mozart-dev.sourceforge.net/xl_status.html)”.

The XL language, like the WSOL language, enables the specification of pre-conditions, post-conditions, and external operation calls to other Web Services. But it is different from WSOL in several ways. XL is not an XML-based language like WSOL. XL does not have any concept such as multiple service offerings for a Web Service. So it

is unable to provide the service providers and consumers with all the benefits resulting from multiple service offerings. It does not enable providing information about access rights (currently), management responsibilities, QoS constraints, price/penalty, etc. It is not currently fully compatible with WSDL.

### **3.2.2.3 Other Works**

In [28], Mckee and Marshall demonstrate how behavioral specification of a software component can be written formally using XML. They show how different constraints such as pre-conditions, post-conditions, invariants, etc. can be specified in XML. They also mention how the performance of a software component can be described in terms of the performance metrics such as response time, scalability indications, etc. in XML. Also, they explain how different policies such as authorization policies can be expressed in XML. This work is less related to WSOL. WSOL also enables specification of constraints such as preconditions, postconditions, QoS constraints, such as performance, and access rights (similar to authorization policies). But unlike WSOL, the work by Mckee and Marshall does not have the concept of multiple service offerings. Also, it does not mention any details about price, penalty, etc.

Another work less related to WSOL is [22] by Jacobsen and Kramer. They develop a framework that enables specification of an IDL and its extensions in an XML-based modeling language. The IDL extensions can include behavioral constraints, such as pre-conditions, post-conditions, invariants, etc. They can also include quality of service constraints, such as execution time, QoS, priorities, deadlines, etc. Moreover, the IDL extensions can also include security constraints, such as access control, authentication

information, etc. The IDL definitions and its extensions are decoupled in order to make the addition of new features easy. The IDL definitions include specification of IDL operations, interfaces etc.

WSOL is little bit similar to this work in that WSOL also enables specification of behavioral constraints, QoS constraints, and access rights. But WSOL is defined to specify these constraints and contracts specifically for Web Services. Also, WSOL has the unique concept of multiple classes of service. WSOL also enables specification of more information such as price, penalty, management responsibilities, etc.

### **3.2.3 Works Least Relevant to WSOL**

#### **3.2.3.1 QML**

QML (Quality of Service Modeling Language) [17] is a language for Quality of Service specification for software components in distributed object systems. QML enables the description of QoS properties such as reliability, performance, etc. The major difference between WSOL and QML is that QML is not an XML-based language like WSOL.

#### **3.2.3.2 DBC**

Another work related to WSOL includes the notion of Design by Contract (DBC) [1], for example in the Eiffel programming language. The idea is to enable components of a software system to communicate with each other according to the mutually agreed upon specifications - contracts. These contracts include specification of pre-conditions, post-conditions, and invariants. These contracts are part of the source code. Several tools in

Java exist that support the DBC method such as iContract, JContract, etc. Unlike WSOL, design by contract does not enable specification of contracts in XML. Also, design by contract does not directly support QoS constraints, access rights, price/penalty statements, etc.

### **3.3 Summary**

The WSOL language and its related works were examined in this chapter. When I started working on my thesis, most of the WSOL-related languages had their corresponding grammars as well as language tools. WSOL did not have any grammar or any tool. Before developing the XML grammar for WSOL, I studied the grammars of the WSOL-related languages. These grammars defined rules for specification of various constraints, quality of service information, etc. in XML. None of these grammars enabled specification of features and concepts similar to those defined in WSOL. Therefore, it was very essential to define a grammar for WSOL language that would enable specification of all the features of WSOL in XML. The following comparison table compares WSOL with the works most relevant to WSOL (i.e., WSLA, WSML, and DAML-S):

<b>Feature</b>	<b>WSOL</b>	<b>WSLA</b>	<b>WSML</b>	<b>DAML-S</b>
<b>Categories of constraints</b>				
<b>Functional constraints</b>	v			v
<b>QoS constraints</b>	v	v	v	v
<b>Access rights</b>	v			
<b>Additional information</b>				
<b>Definition of QoS metrics</b>		v	v	
<b>Action guarantees</b>		v	v	
<b>Management information</b>	v	v	v	
<b>Price/Penalty information</b>	v	v	v	v
<b>Customization of service and QoS</b>				
<b>Multiple classes of service for the same WSDL functionality</b>	v			
<b>Custom-made SLAs</b>		v	v	
<b>Validity period of an SLA</b>		v	v	
<b>Language syntax</b>				
<b>XML-based language</b>	v	v	v	v

<b>Feature</b>	<b>WSOL</b>	<b>WSLA</b>	<b>WSML</b>	<b>DAML-S</b>
<b>Mechanisms for reusing specifications</b>				
<b>Mechanism for grouping specifications</b>	v	v		
<b>Inheritance of specifications</b>	v			
<b>Templates</b>	v	v	v	
<b>Inclusion mechanism</b>	v			
<b>Applications</b>				
<b>Management applications</b>	v	v	v	
<b>Selection of appropriate Web Services</b>	v			v
<b>Tools</b>				
<b>Language tools</b>	v	v	v	v
<b>Management tools</b>		v	v	

**Table 3.1 Comparison between WSOL and the works most relevant to WSOL**

In the above table, the languages WSOL, WSLA, WSML, and DAML-S have been compared in terms of different features they support. Regarding constraints, it has been observed that WSOL enables specification of functional constraints, QoS constraints, and access rights, WSLA enables specification of only QoS constraints, WSML also supports specification of only QoS constraints, while DAML-S supports specification of both functional and QoS constraints. Also, it has been noticed that all the

four languages enable specification of some additional information. WSOL enables specification of management and price/penalty information, WSLA enables specification of action guarantees, management and price/penalty information and also supports definition of QoS metrics, WSML allows specification of action guarantees, management, and price/penalty information, and even supports definition of QoS metrics, while DAML-S only supports specification of price/penalty information.

All the four languages are also compared in terms of service and QoS customization they support. WSOL enables specification of multiple classes of service for the same WSDL functionality, WSLA enables specification of custom-made SLAs and validity period for an SLA, WSML also enables specification of custom-made SLAs and validity period for an SLA, while DAML-S does not support any service and QoS customization. An observation has also been made that all the four languages are XML-based.

Regarding mechanisms for reusing specifications, WSOL supports grouping of specifications, inheritance and inclusion of specifications, and templates. WSLA provides support for both grouping of specifications and templates, WSML supports only templates, while DAML-S does not support any mechanism for reusing specifications. In terms of applications, it has been examined that WSOL, WSLA, and WSML can be used for management purposes, while only WSOL and DAML-S can be used for selection of appropriate Web Services. Regarding tools, all the four languages have basic language tools, while only WSLA and WSML have management tools.

To conclude, from the comparison made in Table 3.1, one can observe that none of the languages related to WSOL is a complete substitution of WSOL. Each of the



WSOL related languages addresses only some of the issues addressed by WSOL. On the other hand, one can also see that WSOL lacks support for several important features that are supported by its related languages.

## 4 WSOL Grammar

---

The grammar for the WSOL language is specified in XML using the XML Schema language [11]. The WSOL grammar is specified in two separate schemas, WSOL Schema and Expression Schema, to improve re-usability. The WSOL Schema consists of many WSOL grammar rules. This chapter discusses only some of the grammar rules (i.e., grammar rules defined for the WSOL root element and the WSOL import element and grammar rules defined for important WSOL concepts such as, constraints, various statements, CG, CGT, and Service Offering) specified in the WSOL Schema. However, the entire WSOL Schema is given in Appendix A.

The Expression Schema specifies the grammar rules for different types of expressions used in WSOL constraints such as, Boolean expressions, arithmetic expressions, arithmeticWithUnit expressions, time expressions, string expressions, and quantified expressions. This chapter does not discuss the grammar rules specified in the Expression Schema, but the complete Expression Schema is given in Appendix B. The grammar rules that are specified in the Expression Schema (using the XML Schema language) were initially specified using the BNF notation [Appendix C]. This BNF notation was developed jointly with Vladimir Tasic, a Ph.D. candidate in our research group.

I have also looked at languages such as XPath [54] and MathML [52] that enable specification of expressions in XML. But, I have not used these languages for specifying WSOL expressions. Instead, I have used XML for specifying expressions in WSOL. There are several reasons behind this choice. MathML is a language developed mainly

for specifying mathematical equations and scientific notations. It does not enable specification of various types of WSOL expressions such as Quantified expressions, ArithmeticWithUnit expressions, string expressions, etc. It also does not enable reference to a WSDL message part from within an expression. Hence, it could not be used for specifying expressions in WSOL. XPath was designed mainly to address parts of an XML document and not for specifying expressions. But it can also be used to specify expressions such as Boolean, String, etc. XPath does not enable specification of some types of WSOL expressions such as Quantified expressions, ArithmeticWithUnit expressions, etc. Moreover, since the syntax of XPath is non-XML, the WSOL parser would have to be modified to process the XPath expression. Hence, the WSOL expressions have been specified in XML.

At the end of this chapter, the grammar developed for specifying dynamic relationships between service offerings is discussed. The buyStock Web Service example (discussed in Section 2.2.2) is used throughout this chapter to explain parts of WSOL Schema.

#### **4.1 WSOL Schema**

The WSOL Schema mainly specifies the grammar for:

- a) The WSOL root element.
- b) The WSOL language constructs constraint, statement, CG, CGT, and service offering, which are all specified within the WSOL root element. These WSOL language constructs were discussed in detail in Section 3.1.

#### 4.1.1 Grammar for the WSOL root element

```
<element name = "WSOLdefinitions" type = "wsol:WSOLdefinitionsType"/>
  <complexType name = "WSOLdefinitionsType">
    <sequence>
      <element ref = "wsol:import" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:externalOperationCall" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:constraint" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:subscription" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:price" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:priceDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:penalty" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:penaltyDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:managementResponsibility" minOccurs = "0"
        maxOccurs = "unbounded"/>
      <element ref = "wsol:statement" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:include" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:CG" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:CGT" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:instantiate" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:serviceOffering" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <any namespace = "##other" processContents = "strict" minOccurs = "0"
        maxOccurs = "unbounded"/>
    </sequence>
    <attribute name = "targetNamespace" type = "anyURI"/>
  </complexType>
```

**Figure 4.1 Grammar for the WSOL root element**

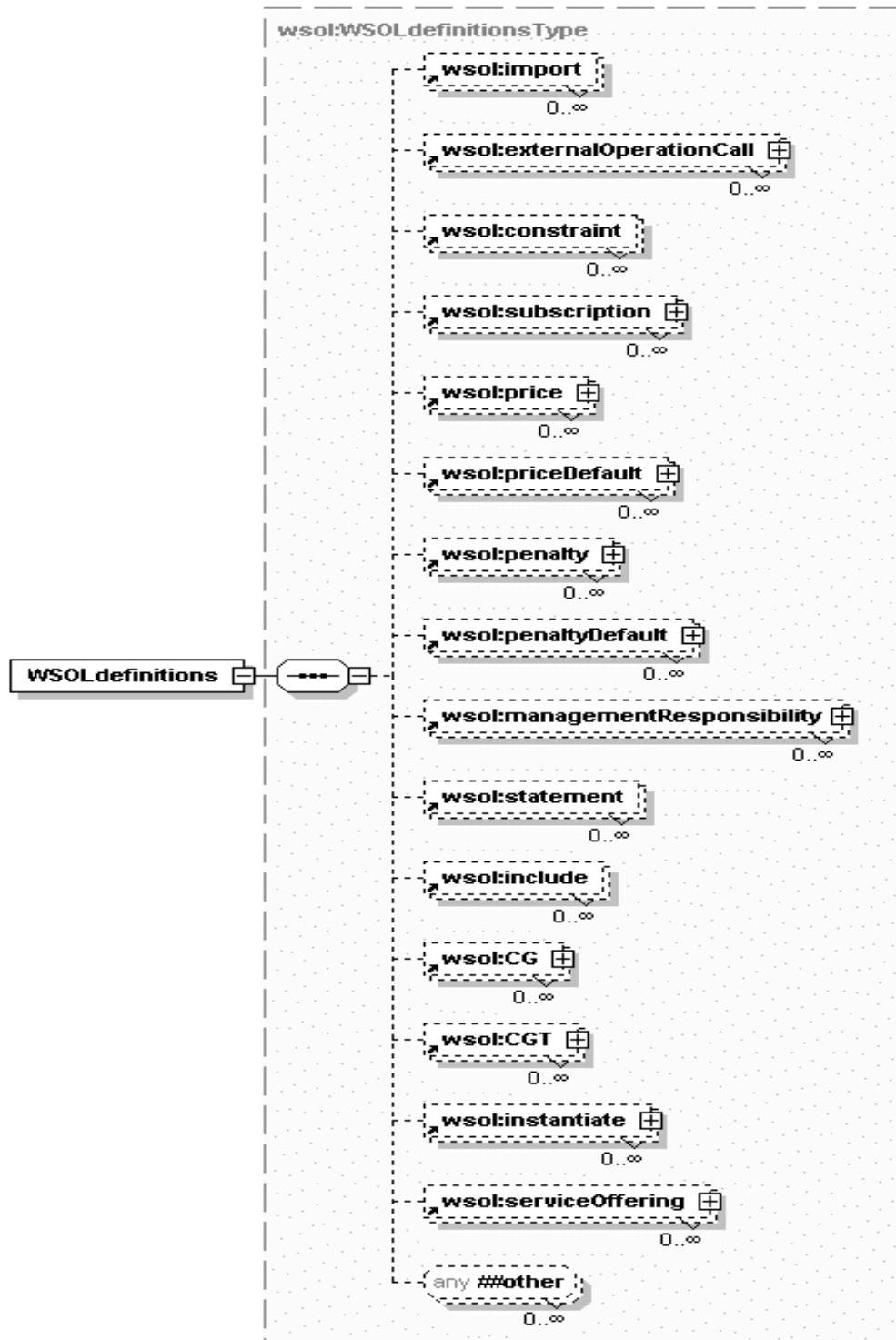


Figure 4.2 Graphical representation of the grammar for the WSOL root element

The WSOL root element, called `<WSOLdefinitions>`, defined in the WSOL Schema, contains definitions of all the other WSOL elements, such as `<constraint>`, `<price>`, `<CG>`, `<CGT>`, `<import>`, etc, defined in the same WSOL document. The `<WSOLdefinitions>` element also contains all the namespace related information for a particular WSOL document, such as default namespace, target namespace, etc. An example of the `<WSOLdefinitions>` element is shown in Figure 4.3:

```

<wsol:WSOLdefinitions          targetNamespace          =
"http://www.sce.carleton.ca/~kpatel/WSOL/serviceOfferings"      xmlns:wsol      =
"http://www.sce.carleton.ca/~kpatel/WSOL/WSOLSchema" ...>
  <wsol:constraint name = "C1"...>...</wsol:constraint>
  <wsol:price name = "PR1"...>...</wsol:price>
  <wsol:CG name = "CG1"...>...</wsol:CG>
  <wsol:CGT name = "CGT1"...>...</wsol:CGT>
  <wsol:serviceOffering name = "SO1"...>...</wsol:serviceOffering>
  ...
</wsol:WSOLdefinitions>

```

**Figure 4.3 Example of a WSOL root element**

In the given example, the `<WSOLdefinitions>` element contains definitions of other WSOL elements, such as a constraint named “C1”, a price statement named “PR1”, a CG named “CG1”, a CGT named “CGT1”, a service offering named “SO1” (defined for the buyStock Web Service), etc. The `<WSOLdefinitions>` element also contains all the namespace related information for the WSOL document for which it is defined.

#### **4.1.2 Grammar for a WSOL Constraint**

As mentioned in Section 3.1.1, the WSOL language enables specification of the following constraints: functional constraints (pre-conditions, post-conditions, and future-conditions), non-functional constraints (i.e., QoS constraints), and access rights. The

general <constraint> element is defined in the WSOL Schema for enabling specification of these different types of constraints. The syntax for the <constraint> element is shown in Figure 4.4:

```
<element name="constraint" type="wsol:constraintType"/>
  <complexType name="constraintType">
    <attribute name="name" use="required" type="NCName"/>
    <attribute name="service" use="required" type="QName"/>
    <attribute name="portOrPortType" use="required" type="QName"/>
    <attribute name="operation" use="required" type="QName"/>
  </complexType>
```

**Figure 4.4 Grammar for a WSOL Constraint**

The <constraint> element enables WSOL specification of the name of the constraint, the domain for which the constraint is specified, the type of the constraint, and the actual constraint expression. The domain for which the constraint is specified is determined by the name of the Web Service, name of its port or a general portType, and operation. Note that special values for the attributes “*service*”, “*portOrPortType*”, and “*operation*” enable very different specification of the domain of a WSOL constraint, as will be discussed later in this chapter.

The type of a constraint can be specified by adding the “*xsi:type*” attribute to the general <constraint> element at the time of specifying the constraint. Consequently, the <constraint> element can be used for the specification of different types of constraints currently supported by WSOL. This also makes the WSOL language extensible by enabling definition of additional types of constraints, currently not supported by WSOL. The value of the “*xsi:type*” attribute is different for different types of constraints. For example, for a pre-condition, the value is “preConditionSchema:preCondition”, while for a QoS constraint the value is “qosSchema:QoSconstraint”. Each type of constraint is

defined in its corresponding schema. For example, the constraint type “preCondition” is defined in the Schema “PreConditionSchema”.

Let us now look at an example of a constraint specified in WSOL for the buyStock Web Service:

```
<wsol:constraint      name="C3"      xsi:type="preConditionSchema:preCondition"
service="buyStock:buyStockService"  portOrPortType="buyStock:buyStockServicePort"
operation="buyStock:buySingleStockOperation">
  <expressionSchema:booleanExpression>
    <expressionSchema:arithmeticExpression>
      <expressionSchema:arithmeticVariable          avName=
        buyStock:buySingleStockRequest.quantity"/>
    </expressionSchema:arithmeticExpression>
    <expressionSchema:arithmeticComparator type=">"/>
    <expressionSchema:arithmeticExpression>
      <expressionSchema:arithmeticConstant>
        <expressionSchema:integerConstant value="0"/>
      </expressionSchema:arithmeticConstant>
    </expressionSchema:arithmeticExpression>
  </expressionSchema:booleanExpression>
</wsol:constraint>
```

**Figure 4.5 Example of a WSOL Constraint**

The example constraint is of type “pre-condition”. It is defined for a particular operation and for a particular port of the buyStock Web Service. The shown constraint expression specifies a pre-condition that the quantity of stocks to be bought should always be greater than zero.

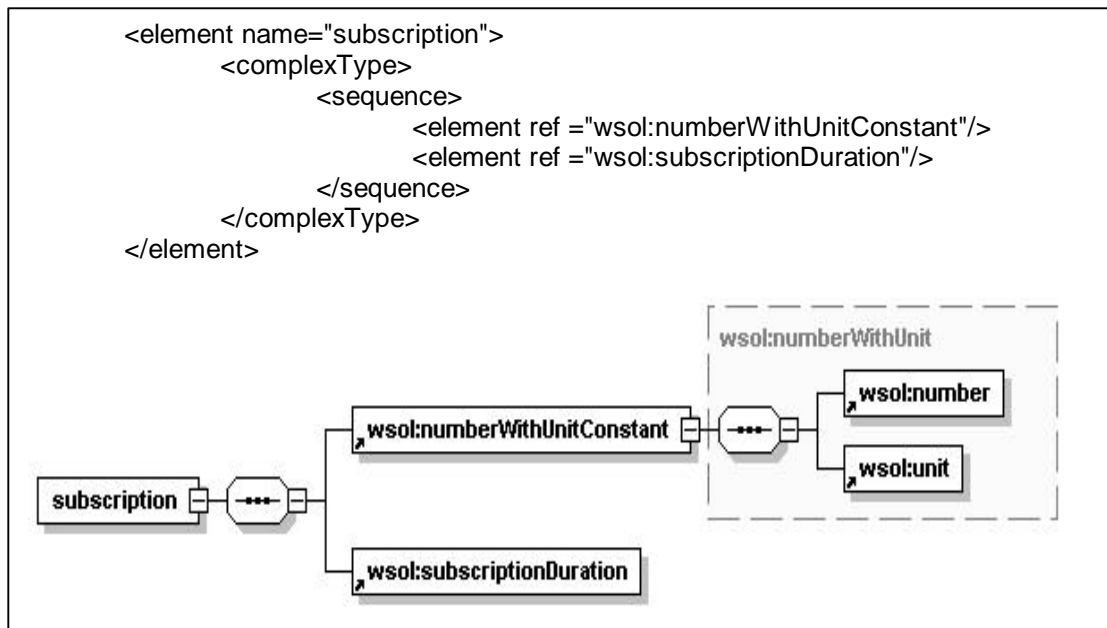
#### **4.1.3 Grammar for a WSOL Statement**

As stated in Section 3.1.2, the WSOL language enables specification of the following statements: the subscription statement, the price statement, the price default statement, the penalty statement, the penalty default statement, the external operation call statement, the management responsibility statement, the Include statement, and the Instantiate



statement. The grammar for each type of statement is different. Therefore, a specific element is defined in the WSOL Schema for specifying each type of statement. The syntax for each type of statement is discussed in the following subsections. The grammar for some of the statements (e.g., subscription, price, etc.) references an element called <numberWithUnitConstant>. This element is defined in the WSOL Schema and it enables specification of a constant of data type “numberWithUnit”. The data type “numberWithUnit” is also defined in the WSOL Schema and it consists of a numeric constant followed by a measurement unit such as “millisecond”. An example of a constant of data type “numberWithUnit” would be “5 millisecond”.

#### 4.1.3.1 Grammar for a WSOL Subscription Statement



**Figure 4.6 Grammar for a WSOL Subscription Statement**

The <subscription> element is defined in the WSOL Schema to enable specification of a subscription statement. It enables specification of:

- 1) the subscription price, using the element <numberWithUnitConstant>.
- 2) the subscription duration, using the element <subscriptionDuration>.

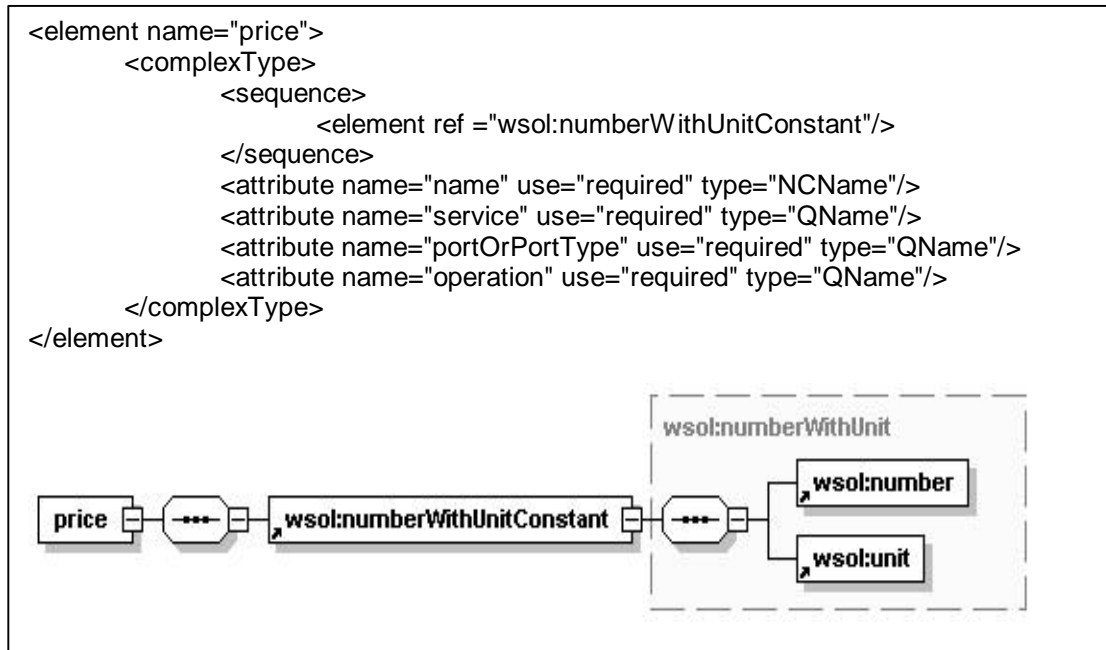
An example of a subscription statement specified in WSOL for the buyStock Web Service is given in Figure 4.7:

```
<wsol:subscription>  
  <wsol:numberWithUnitConstant>  
    <wsol:number value="5"/>  
    <wsol:unit type="currencyOntology:Dollar"/>  
  </wsol:numberWithUnitConstant>  
  <wsol:subscriptionDuration duration="P1Y"/>  
</wsol:subscription>
```

**Figure 4.7 Example of a WSOL Subscription Statement**

This example statement specifies that the subscription price to be paid by the buyStock Web Service consumer to the buyStock Web Service supplier is 5 Dollar and the subscription duration is 1 year.

#### 4.1.3.2 Grammar for a WSOL Price Statement



**Figure 4.8 Grammar for a WSOL Price Statement**

The <price> element is defined in the WSOL Schema to enable specification of a price statement. The price is specified as a <numberWithUnitConstant>. The <price> element also enables specification of the name of the price statement and the domain for which the price statement is specified. An example of a price statement specified in WSOL for the buyStock Web Service is given in Figure 4.9:

```

<wsol:price      name="Price1"      service="buyStock:buyStockService"
portOrPortType="buyStock:buyStockServicePort"      operation=
buyStock:buySingleStockOperation">
  <wsol:numberWithUnitConstant>
    <wsol:number value="3"/>
    <wsol:unit type="currencyOntology:Dollar"/>
  </wsol:numberWithUnitConstant>
</wsol:price>

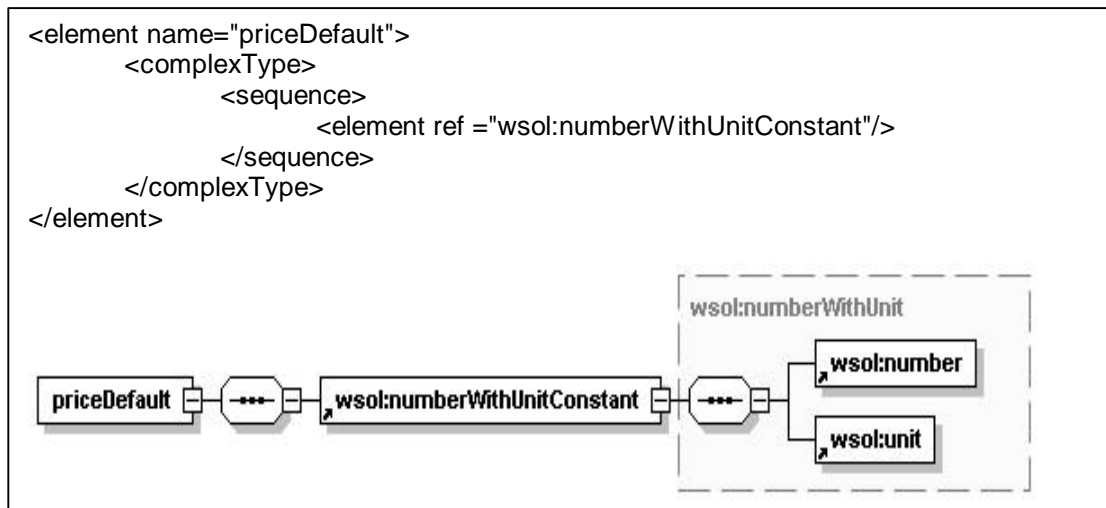
```

**Figure 4.9 Example of a WSOL Price Statement**

This price statement is defined for a particular operation and for a particular port of the buyStock Web Service. It specifies that the price to be paid by the buyStock Web Service consumer to the buyStock Web Service supplier is 3 Dollar.

The grammar for a WSOL penalty statement is the same as the grammar for a WSOL price statement. So, the grammar for a WSOL penalty statement is not discussed explicitly in this chapter.

#### 4.1.3.3 Grammar for a WSOL Price Default Statement



**Figure 4.10 Grammar for a WSOL Price Default Statement**

The <priceDefault> element is defined in the WSOL Schema to enable specification of a price default statement. The default price is specified as a <numberWithUnitConstant>. An example of a price default statement specified in WSOL for the buyStock Web Service is given in Figure 4.11:

```

<wsol:priceDefault>
  <wsol:numberWithUnitConstant>
    <wsol:number value="2"/>
    <wsol:unit type="currencyOntology:Dollar"/>
  </wsol:numberWithUnitConstant>
</wsol:priceDefault>

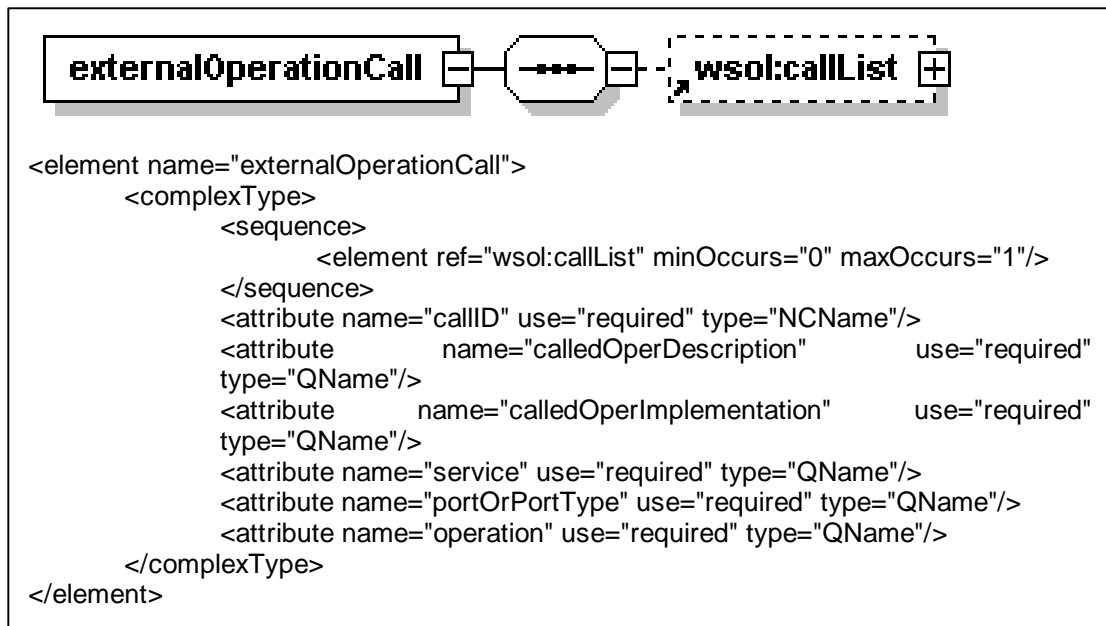
```

**Figure 4.11 Example of a WSOL Price Default Statement**

This statement specifies that the default price to be paid by the buyStock Web Service consumer to the buyStock Web Service supplier is 2 Dollar.

The grammar for a WSOL penalty default statement is the same as the grammar for a WSOL price default statement. So, the grammar for a WSOL penalty default statement is not discussed explicitly in this chapter.

#### 4.1.3.4 Grammar for a WSOL External Operation Call Statement



**Figure 4.12 Grammar for a WSOL External Operation Call Statement**

The <externalOperationCall> element is defined in the WSOL Schema to enable specification of an external operation call statement. It enables specification of the external operation's input parameters and their corresponding values, using element <callList>. The <externalOperationCall> element also enables specification of the following information:

- id of the external operation call statement,
- description of the external operation being called - the name of the external operation and the name of the port type where it is defined,
- implementation address of the external operation being called - the name of the port that implements the called external operation, and
- the domain for which the external operation call is made.

An example of an external operation call statement specified in WSOL for the buyStock Web Service is given in Figure 4.13:

```

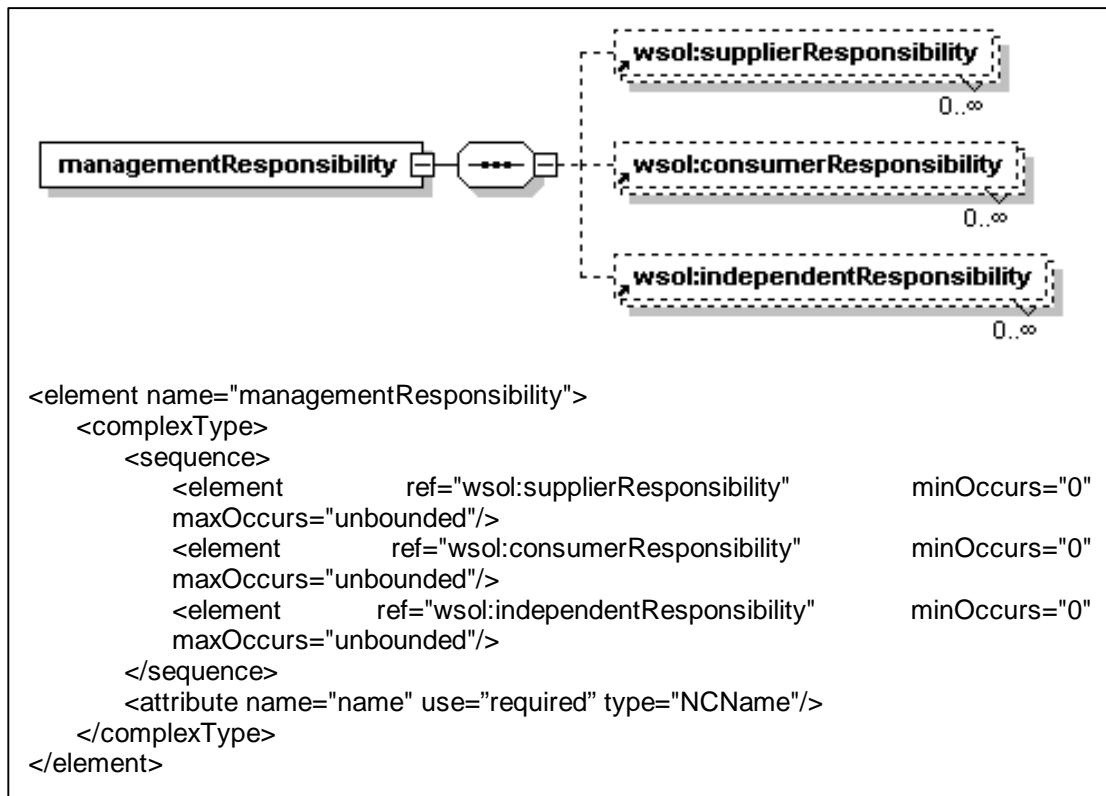
<wsol:externalOperationCall callID = "extOp1" calledOperDescription =
"symbolLookup:symbolLookupPortType.symbolExists" calledOperImplementation =
"symbolLookup:symbolLookupService.symbolLookupServicePort" service =
"buyStock:buyStockService" portOrPortType = "buyStock:buyStockServicePort" operation
= "buyStock:buySingleStockOperation">
  <wsol:callList>
    <wsol:parameterValue>
      <wsol:partName
name="symbolLookup:singleSymbolLookupRequest.symbol"/>
      <wsol:paramValue>
        <expressionSchema:stringExpression>
          <expressionSchema:stringVariable svName =
"buyStock:buySingleStockRequest.symbol"/>
        </expressionSchema:stringExpression>
      </wsol:paramValue>
    </wsol:parameterValue>
  </wsol:callList>
</wsol:externalOperationCall>

```

**Figure 4.13 Example of a WSOL External Operation Call Statement**

In this example, the name of the external operation being called is “symbolExists”. This external operation is provided by the symbolLookup Web Service and is called when a consumer of the buyStock Web Service invokes an operation “buySingleStockOperation”, implemented at the port “buyStockServicePort” of the buyStock Web Service. The external operation verifies whether a symbol of a company given by the consumer exists or not.

#### 4.1.3.5 Grammar for a WSOL Management Responsibility Statement



**Figure 4.14 Grammar for a WSOL Management Responsibility Statement**

The <managementResponsibility> element is defined in the WSOL Schema to enable specification of a management responsibility statement. It enables specification of:

- the scope of management responsibility of the Web Service supplier, using element <supplierResponsibility>,
- the scope of management responsibility of the Web Service consumer, using element <consumerResponsibility>,
- the scope of management responsibility of one or more independent third parties trusted by the supplier and the consumer, using element <independentResponsibility>,
- the name of the management responsibility statement, using the attribute “name”.

The scope of management responsibility can be a particular constraint, a constraint group (CG) inside a service offering, or a complete service offering. An example of a management responsibility statement specified in WSOL for the buyStock Web Service is given in Figure 4.15:

```

<wsol:managementResponsibility name="MangResp1">
  <wsol:supplierResponsibility scope="tns:AccRghtCons1"/>
  <wsol:consumerResponsibility scope="tns:C3"/>
  <wsol:independentResponsibility scope="tns:QoScons1"
  entity="http://www.thirdParty.com"/>
</wsol:managementResponsibility>

```

**Figure 4.15 Example of a WSOL Management Responsibility Statement**

According to this statement, the buyStock Web Service supplier has the management responsibility for checking an access right constraint named “AccRghtCons1”, the buyStock Web Service consumer has the management responsibility for checking a functional constraint named “C3”, and an independent third party (denoted by uri - "http://www.thirdParty.com") has the management responsibility for checking a QoS constraint named “QoScons1”.



#### 4.1.3.6 Grammar for a WSOL Include Statement

```
<element name = "include">
  <complexType>
    <attribute name = "constructName" use = "required" type = "QName"/>
    <attribute name = "resService" use = "required" type = "QName"/>
    <attribute name = "resPortOrPortType" use = "required" type =
      "QName"/>
    <attribute name = "resOperation" use = "required" type = "QName"/>
    <attribute name = "resName" use = "required" type = "NCName"/>
  </complexType>
</element>
```

**Figure 4.16 Grammar for a WSOL Include Statement**

The <include> element is defined in the WSOL Schema to enable specification of an include statement. It enables specification of:

- the name of the WSOL construct to be included,
- the new domain of the included WSOL construct, and
- the new name of the included WSOL construct.

An example of an include statement specified in WSOL for the buyStock Web Service is given in Figure 4.17:

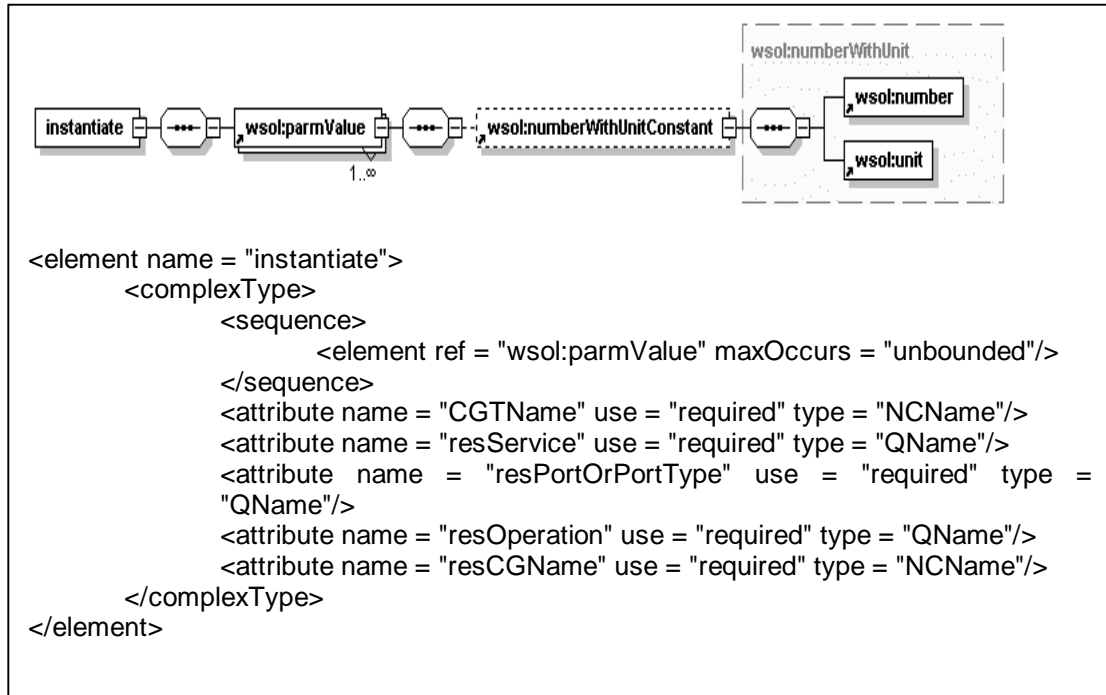
```
<wsol:include constructName="C3" resService="buyStock:buyStockService"
resPortOrPortType="buyStock:buyStockServicePort"
resOperation="buyStock:buySingleStockOperation" resName="NewC3"/>
```

**Figure 4.17 Example of a WSOL Include Statement**

According to this statement, the WSOL construct being included is a constraint named “C3”. After being included, the constraint will be named as “NewC3”. The new

domain of the included constraint is the buySingleStockOperation of the buyStockServicePort of the buyStock Web Service.

#### 4.1.3.7 Grammar for a WSOL Instantiate Statement



**Figure 4.18 Grammar for a WSOL Instantiate Statement**

The <instantiate> element is defined in the WSOL Schema to enable specification of an instantiate statement. It enables specification of:

- the name of the constraint group template (CGT) to be instantiated,
- the domain to which the instantiated CGT will apply,
- the name of the constraint group (CG) that will be the result of the CGT instantiation,
- value of each parameter of the CGT being instantiated, using element <parmValue>

An example of an instantiate statement specified in WSOL for the buyStock Web Service is given in Figure 4.19:

```
<wsol:instantiate CGTName="CGT2" resService="buyStock:buyStockService"
resPortOrPortType="buyStock:buyStockServicePort"
resOperation="buyStock:buySingleStockOperation" resCGName="CG5">
  <wsol:parmValue name = "responseTime">
    <wsol:numberWithUnitConstant>
      <wsol:number value="30"/>
      <wsol:unit type = "QoSMeasOntology:millisecond"/>
    </wsol:numberWithUnitConstant>
  </wsol:parmValue>
</wsol:instantiate>
```

**Figure 4.19 Example of a WSOL Instantiate Statement**

This statement instantiates a CGT named “CGT2” by providing a concrete value (30 millisecond) for the parameter “responseTime” of the CGT. After instantiation, the CGT becomes a CG named “CG5”. The domain of this new CG is the buySingleStockOperation of the buyStockServicePort of the buyStock Web Service.

To improve extensibility of WSOL, the general <statement> element, similar to the general <constraint> element, is also defined in the WSOL Schema. This <statement> element could be used to specify additional types of statements, currently not supported by WSOL. The syntax for the <statement> element is as follows:

```
<element name = "statement">
  <complexType>
    <attribute name = "name" use="required" type = "NCName"/>
  </complexType>
</element>
```

**Figure 4.20 Grammar for a general WSOL Statement**

The <statement> element currently enables specification of the name of the statement, the type of the statement, and the statement expression. The type of a statement can be specified by adding the “*xsi:type*” attribute to the general <statement> element at the time of specifying the statement. Consequently, the <statement> element can be used for specifying different types of statements.

#### 4.1.4 Grammar for a WSOL Constraint Group (CG)

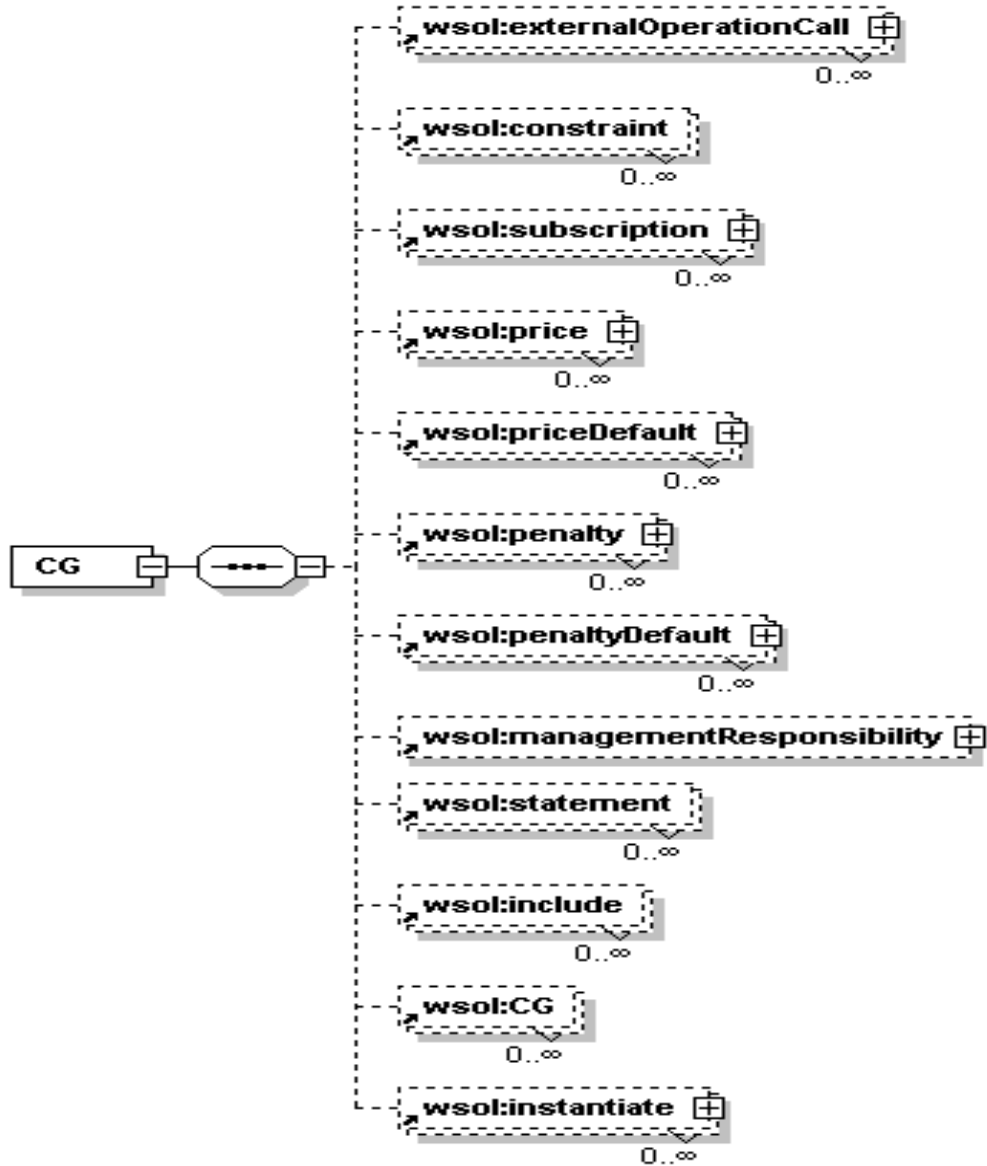
```

<element name = "CG">
  <complexType>
    <sequence>
      <element ref = "wsol:externalOperationCall" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:constraint" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:subscription" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:price" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:priceDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:penalty" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:penaltyDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:managementResponsibility" minOccurs = "0"/>
      <element ref = "wsol:statement" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:include" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:CG" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:instantiate" minOccurs = "0" maxOccurs =
        "unbounded"/>
    </sequence>
    <attribute name = "name" use = "required" type = "NCName"/>
    <attribute name = "extends" use = "optional" default = "WSOL-NONE" type =
      "QName"/>
    <attribute name = "service" use = "required" type = "QName"/>
    <attribute name = "portOrPortType" use = "required" type = "QName"/>
    <attribute name = "operation" use = "required" type = "QName"/>
  </complexType>
</element>

```

**Figure 4.21 Grammar for a WSOL Constraint Group (CG)**

A graphical representation of the grammar for a WSOL constraint group is given in Figure 4.22:



**Figure 4.22 Graphical representation of the grammar for a WSOL constraint group**

The element <CG> is defined in the WSOL Schema to enable specification of a constraint group. It enables specification of:

- one or more constraints within a constraint group, using the element <constraint>,

- different types of statements within a constraint group, using the appropriate elements,
- one or more new constraint groups within a constraint group, using the element <CG>.

The element <CG> also enables specification of the name of the constraint group, the name of an existing CG from which the current CG extends (using the attribute “*extends*”), and the domain for which the constraint group is specified. The attribute “*extends*” is optional. If this attribute is not specified explicitly, then it is assumed that the current CG does not extend any existing CG and the value of the attribute “*extends*” becomes “WSOL-NONE”. An example of a constraint group specified in WSOL for the buyStock Web Service is given in Figure 4.23:

```

<wsol:CG name="CG7" service="buyStock:buyStockService" portOrPortType=
buyStock:buyStockServicePort" operation="buyStock:buyMultipleStockOperation">
  <wsol:constraint name = "C7" xsi:type = "..." service = "..." portOrPortType = "..."
operation = "...">...</wsol:constraint>
  <wsol:subscription>...</wsol:subscription>
  <wsol:managementResponsibility name = "MR1">...</wsol:managementResponsibility>
  <wsol:CG name = "CG8" service = "..." portOrPortType = "..." operation = "...">
  ...
  </wsol:CG>
  ...
</wsol:CG>

```

**Figure 4.23 Example of a WSOL constraint group**

In the above example, the CG named “CG7” is defined that contains a constraint named “C7”, a subscription statement, a management responsibility statement named “MR1”, and a new CG named “CG8”.

#### 4.1.5 Grammar for a WSOL Constraint Group Template (CGT)

The grammar for a WSOL constraint group template (CGT) is the same as the grammar for a WSOL constraint group (CG) (discussed in the previous subsection) except that the grammar for a WSOL CGT contains an additional statement “<element ref = "wsol:parameter" minOccurs = "1" maxOccurs = "unbounded"/>” to enable specification of one or more CGT parameters.

The <parameter> element enables specification of the name of the CGT parameter, the data type of the CGT parameter, and the measurement unit of the CGT parameter. The specification of the measurement unit of the CGT parameter is optional. It is specified explicitly only if the data type of the CGT parameter is “numberWithUnit”. Otherwise, its default value is set to the constant “WSOL-NOUNIT”. An example of a CGT specified in WSOL for the buyStock Web Service is given in Figure 4.24:

```
<wsol:CGT name = "CGT2" service = "buyStock:buyStockService" portOrPortType =
"buyStock:buyStockServicePort" operation = "buyStock:buySingleStockOperation">
  <wsol:parameter name = "responseTime" dataType = "wsol:numberWithUnit"
unit="QoSMeasOntology:millisecond"/>
  <wsol:constraint name = "QoScons2" xsi:type = "qosSchema:QoSconstraint"
service = "..." portOrPortType = "..." operation = "...">
<expressionSchema:booleanExpression>
  <expressionSchema:arithmeticWithUnitExpression>
    <expressionSchema:QoSmetric metricType =
"QoSmetricOntology:ResponseTime" service = "..."
portOrPortType = "..." operation = "..." measuredBy = "..."/>
  </expressionSchema:arithmeticWithUnitExpression>
  <expressionSchema:arithmeticWithUnitComparator type = "&lt;"/>
  <expressionSchema:arithmeticWithUnitExpression>
    <expressionSchema:arithmeticWithUnitVariable aWUName =
"tns:CGT2.responseTime"/>
  </expressionSchema:arithmeticWithUnitExpression>
</expressionSchema:booleanExpression>
  </wsol:constraint>
  ...
</wsol:CGT>
```

Figure 4.24 Example of a WSOL constraint group template

In the above example, a new CGT named “CGT2” is defined that contains a CGT parameter named “responseTime” of data type “numberWithUnit” and measurement unit “millisecond”. It also contains a QoS constraint, in which a comparison between a QoS metric “ResponseTime” and the CGT parameter “responseTime” is specified.

#### 4.1.6 Grammar for a WSOL Service Offering

```

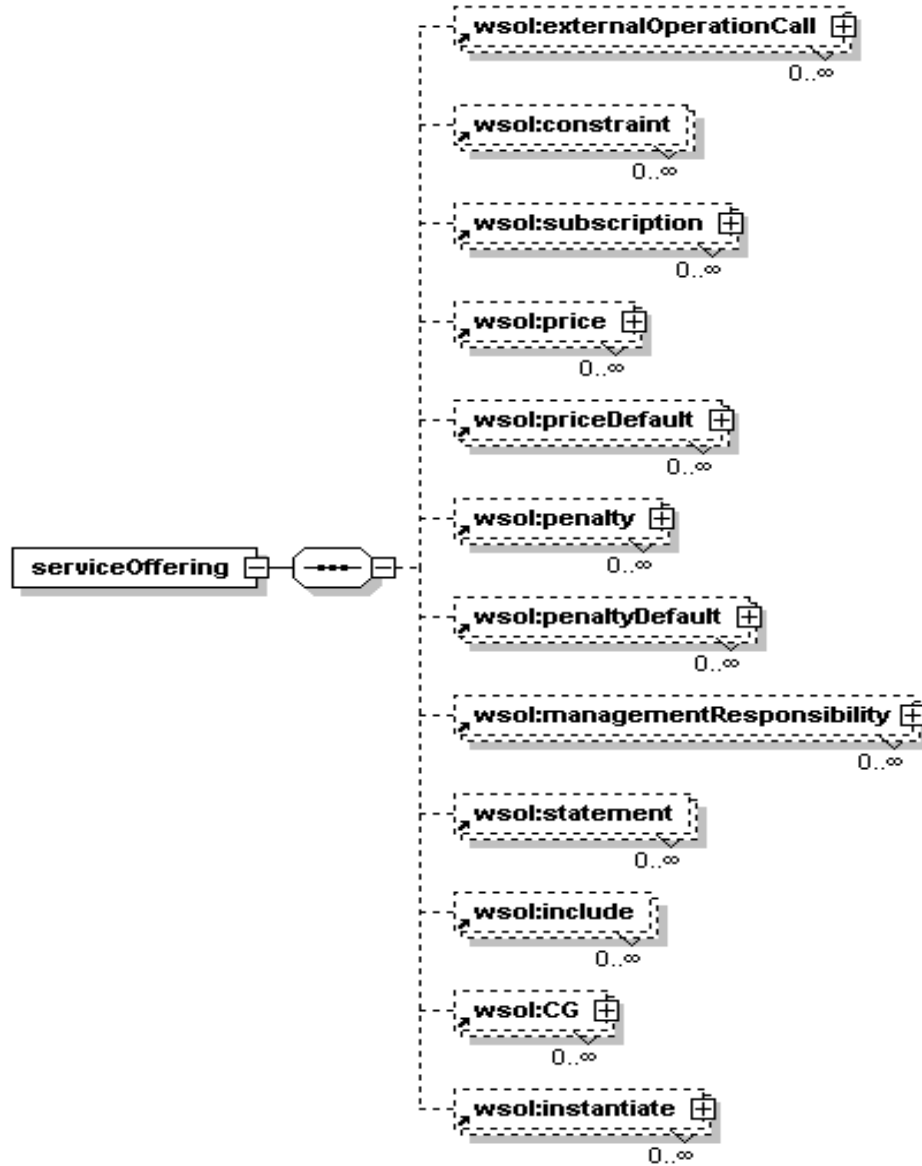
<element name = "serviceOffering">
  <complexType>
    <sequence>
      <element ref = "wsol:externalOperationCall" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:constraint" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:subscription" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:price" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:priceDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:penalty" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:penaltyDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:managementResponsibility" minOccurs = "0"
        maxOccurs = "unbounded"/>
      <element ref = "wsol:statement" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:include" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:CG" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:instantiate" minOccurs = "0" maxOccurs =
        "unbounded"/>
    </sequence>
    <attribute name = "name" use = "required" type = "NCName"/>
    <attribute name = "service" use = "required" type = "QName"/>
    <attribute name = "extends" use = "optional" default = "WSOL-NONE" type =
      "QName"/>
    <attribute name = "accountingParty" use = "optional" default = "WSOL-
      SUPPLIERWS" type = "anyURI"/>
  </complexType>
</element>

```

**Figure 4.25 Grammar for a WSOL Service Offering**



A graphical representation of the grammar for a WSOL service offering is given in Figure 4.26:



**Figure 4.26 Graphical representation of the grammar for a WSOL service offering**

The element <serviceOffering> is defined in the WSOL Schema to enable specification of a service offering. It enables specification of:

- one or more constraints within a service offering, using the element <constraint>,

- different types of statements within a service offering, using the appropriate elements,
- one or more constraint groups within a service offering, using the element <CG>.

The element <serviceOffering> also enables specification of the name of the service offering, the name of the Web Service for which the service offering is specified, the name of an existing service offering from which the current service offering extends (using the attribute “*extends*”), and the name of the accounting party (a special management party that has the management responsibility for all price/penalty statements) for the service offering. The specification of the attribute “*extends*” is optional. If this attribute is not specified explicitly, then it is assumed that the current service offering does not extend any existing service offering and the value of the attribute “*extends*” becomes “WSOL-NONE”. Similarly, the specification of the attribute “*accountingParty*” is optional. If it is not specified explicitly, then it is assumed that the supplier Web Service is the accounting party (denoted by “WSOL-SUPPLIERWS”). Otherwise, the accounting party can be a consumer (denoted by “WSOL-CONSUMERWS”) or some third party. An example of a service offering specified in WSOL for the buyStock Web Service is given in Figure 4.27:

```

<wsol:serviceOffering name = "SO1" service = "buyStock:buyStockService"
accountingParty = "WSOL-CONSUMERWS">
  <wsol:constraint name = "C2" xsi:type = "preConditionSchema:preCondition"
service = "buyStock:buyStockService" portOrPortType =
"buyStock:buyStockServicePort" operation =
"buyStock:buySingleStockOperation">...</wsol:constraint>
  <wsol:price>...</wsol:price>
  <wsol:managementResponsibility name =
"MR2">...</wsol:managementResponsibility>
  <wsol:include constructName = "C3">...</wsol:include>
  <wsol:CG name = "CG10" service = "..." portOrPortType = "..." operation = "...">
  ...
</wsol:CG>
  ...
</wsol:serviceOffering>

```

**Figure 4.27 Example of a WSOL service offering**

In the above example, a new service offering “SO1” is defined. “SO1” contains a constraint named “C2”, a price statement, a management responsibility statement named “MR2”, an include statement that includes a constraint named “C3”, and a new CG named “CG10”.

Besides specifying the grammar for the WSOL root element and the WSOL constructs, the WSOL Schema also specifies the grammar for the WSOL import mechanism. The WSOL import mechanism enables a WSOL document to import other WSOL documents. This increases the modularity and reusability of WSOL documents. The grammar for the WSOL import mechanism is shown in Figure 4.28:

```

<element name = "import" type = "wsol:importType"/>
  <complexType name = "importType">
    <attribute name = "namespace" use = "required" type = "anyURI"/>
    <attribute name = "location" use = "required" type = "anyURI"/>
  </complexType>

```

**Figure 4.28 Grammar for the WSOL import mechanism**

The element <import> is defined in the WSOL Schema to enable a WSOL document to import other WSOL documents. It enables specification of the namespace and the location of the WSOL document being imported.

## “ Rules for Domain Attributes used in WSOL Constructs

The attributes “*service*”, “*portOrPortType*”, and “*operation*” are referred to in WSOL as domain attributes. The domain attributes, when specified for a WSOL construct, determine the domain of the WSOL construct. In other words, they determine the service, the port or port type, and the operation for which the WSOL construct is specified. The three domain attributes are specified for every constraint, price/penalty statement, external operation call statement, CG, CGT, and QoS metric. For a service offering, only the domain attribute “*service*” is specified, because a service offering contains constraints for all ports and operations of a Web Service. For Include and Instantiate statements, “res” (means result) is added in front of the name of each domain attribute to form attributes “*resService*”, “*resPortOrPortType*”, and “*resOperation*”.

Several constants have been defined in WSOL that can be used as values of the domain attributes:

- WSOL-ANY

WSOL-ANY denotes that the WSOL construct can be applied to “any” service, port or port type, and/or operation. A WSOL construct with such domain is specified outside service offerings and can be included (in the case of CGTs – instantiated) in many different service offerings. However, after the inclusion or

CGT instantiation, the result domain must not have any “WSOL-ANY” value. The constant WSOL-ANY can be used for domain attributes of every constraint, statement, CG, CGT, and QoS metric. But, it cannot be used for the domain attribute “*service*” of the service offering construct. This is because a service offering must be specified for a particular Web Service. The following rules apply for using the constant “WSOL-ANY” in domain attributes:

- When the value of the domain attribute “*service*” is “WSOL-ANY”, then the value of the domain attribute “*portOrPortType*” refers to a particular port type, but when the value of the domain attribute “*service*” is a particular service, then the value of the domain attribute “*portOrPortType*” refers to a particular port of this service.
- When the value of the domain attribute “*portOrPortType*” is “WSOL-ANY”, then the value of the domain attribute “*operation*” must be “WSOL-ANY”.
- WSOL-EVERY

“WSOL-EVERY” denotes that the WSOL construct is applied to “every” service, port or port type, and/or operation. For example, if the value of the “*operation*” attribute of a QoS constraint is “WSOL-EVERY”, this means that this QoS constraint, although specified only once, should be applied to every operation inside the particular port of the particular service. The constant WSOL-EVERY can be used for domain attributes of every constraint, statement, CG, CGT, and QoS metric. However, it cannot be used for the domain attribute “*service*” of the service offering construct. The following rule applies for using the constant “WSOL-EVERY” in domain attributes:

➤ When the value of the domain attribute “*portOrPortType*” is “WSOL-  
EVERY”, then the value of the domain attribute “*operation*” must be  
“WSOL-EVERY”.

- WSOL-MANY

The constant WSOL-MANY can be used only for domain attributes “*portOrPortType*” and “*operation*” of a CG or a CGT. It cannot be used for domain attributes of other constructs, including the domain attribute “*service*” of the service offering construct. WSOL-MANY, when used as a value of the “*operation*” domain attribute, denotes that the CG or CGT contains constraints whose domains are different operations within the same port type, port, or service. Similarly, when WSOL-MANY is used as a value of the “*portOrPortType*” domain attribute, it denotes that the CG or CGT contains constraints whose domains are different ports within the same service. The following rule applies for using the constant “WSOL-MANY” in domain attributes:

➤ When the value of the domain attribute “*portOrPortType*” is “WSOL-  
MANY”, then the value of the domain attribute “*operation*” must be  
“WSOL-MANY”.

- WSOL-ALL

The constant “WSOL-ALL” can be used only for domain attributes “*portOrPortType*” and “*operation*” of QoS metrics. It cannot be used for domain attributes of other constructs, including the domain attribute “*service*” of the service offering construct. Specifying the constant “WSOL-ALL” denotes that a

QoS metric is calculated using all invocations of all operations in a port type, port, or service.

- When the value of the domain attribute “*portOrPortType*” is “WSOL-ALL”, then the value of the domain attribute “*operation*” must be “WSOL-ALL”.
- When the value of the domain attribute “*service*” is “WSOL-ANY”, then the value of the domain attribute “*portOrPortType*” must NOT be “WSOL-ALL”.

A WSOL construct is referred to as an “inner WSOL construct” when it is contained within another WSOL construct, and as an “outer WSOL construct” when it contains another WSOL construct. The domain of the inner WSOL construct is referred to as the “inner domain”, while the domain of the outer WSOL construct is referred to as the “outer domain”. In WSOL, an inner domain must be a sub-domain of its outer domain. In order to check whether an inner domain is a sub-domain of its outer domain, the following rule should be applied to every domain attribute (“*service*”, “*portOrPortType*”, and “*operation*”) in the inner domain and the outer domain: “IF the value of a domain attribute in the outer domain is NOT “WSOL-MANY”, THEN the value of the same domain attribute in the inner domain MUST be the same as in the outer domain; ELSE the same domain attribute in the inner domain CAN have any value (including the constants “WSOL-ANY”, “WSOL-MANY”, and “WSOL-EVERY”)”.

In Include and Instantiate statements, the WSOL construct included or instantiated already has its domain defined. This domain is referred to as the “old

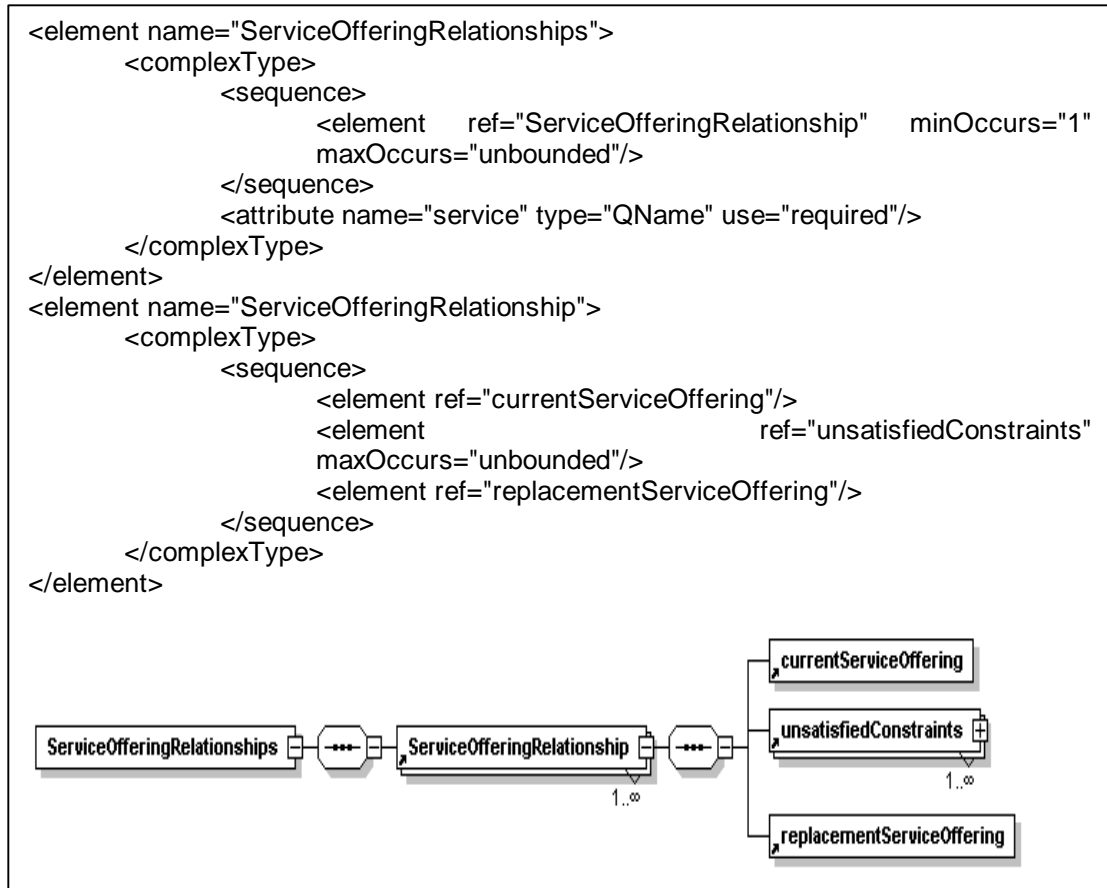
domain”. After being included or instantiated, the WSOL construct has a new domain defined by the attributes “*resService*”, “*resPortOrPortType*”, and “*resOperation*”. This is referred to as the “new domain”. In WSOL, a new domain must be either same as the old domain or a specialization of the old domain. In order to determine this, the following two rules should be checked in the given order:

1. IF in the old domain it is: *service*=”WSOL-ANY” AND NOT *portOrPortType*=”WSOL-ANY” (i.e., if the attribute “*portOrPortType*” refers to the name of some port type), THEN in the new domain the value of attributes “*resService*” and “*resPortOrPortType*” CAN be such to point to a particular service and its port of the port type specified for the old domain.
2. IF a value of an attribute in the old domain is NOT “WSOL-ANY”, THEN the value of the corresponding attribute in the new domain MUST be the same, ELSE the value of the corresponding attribute in the new domain CAN be either “WSOL-ANY” or a specific value that makes the new domain exist.

## **4.2 Grammar for the Specification of Dynamic Relationships between Service Offerings**

As discussed in Section 3.1, dynamic relationships between service offerings can be specified outside WSOL files in a special XML format. The syntax for this specification is shown in Figure 4.29:





**Figure 4.29 Grammar for the Specification of Dynamic Relationships between Service Offerings**

Consider the specification of dynamic relationships between service offerings for the buyStock Web Service shown below:

```

<serviceOfferingRelationships service = "buyStock:buyStockService">
  <serviceOfferingRelationship>
    <currentServiceOffering name = "serviceOfferings:SO1"/>
    <unsatisfiedConstraints>
      <unsatisfied name = "Constraint3"/>
      <unsatisfied name = "CG5"/>
      ...
    </unsatisfiedConstraints>
    <replacementServiceOffering name = "serviceOfferings:SO2"/>
  </serviceOfferingRelationship>
</serviceOfferingRelationships>

```

**Figure 4.30 Example of a specification of Dynamic Relationships between Service Offerings**

The above specification states that if the service offering “SO1” has a set of constraints that are not satisfied (including “Constraint3”, “CG5”, etc.), then the consumers of the service offering “SO1” can switch to using the service offering “SO2”.

### 4.3 Summary

Significant parts of the developed WSOL grammar specified in the WSOL Schema were discussed in this chapter. The grammar rules for the WSOL root element, the WSOL import element and for important WSOL concepts such as, constraints, various statements, CG, CGT, and Service Offering were presented along with a graphical representation of each rule using the XML Spy tool, and an explanation and a WSOL example for each grammar rule. The rules for the domain attributes “*service*”, “*portOrPortType*”, and “*operation*” used in WSOL constructs were discussed. Finally, the grammar for specification of dynamic relationships between Service Offerings was also discussed.

## 5 The WSOL Parser

---

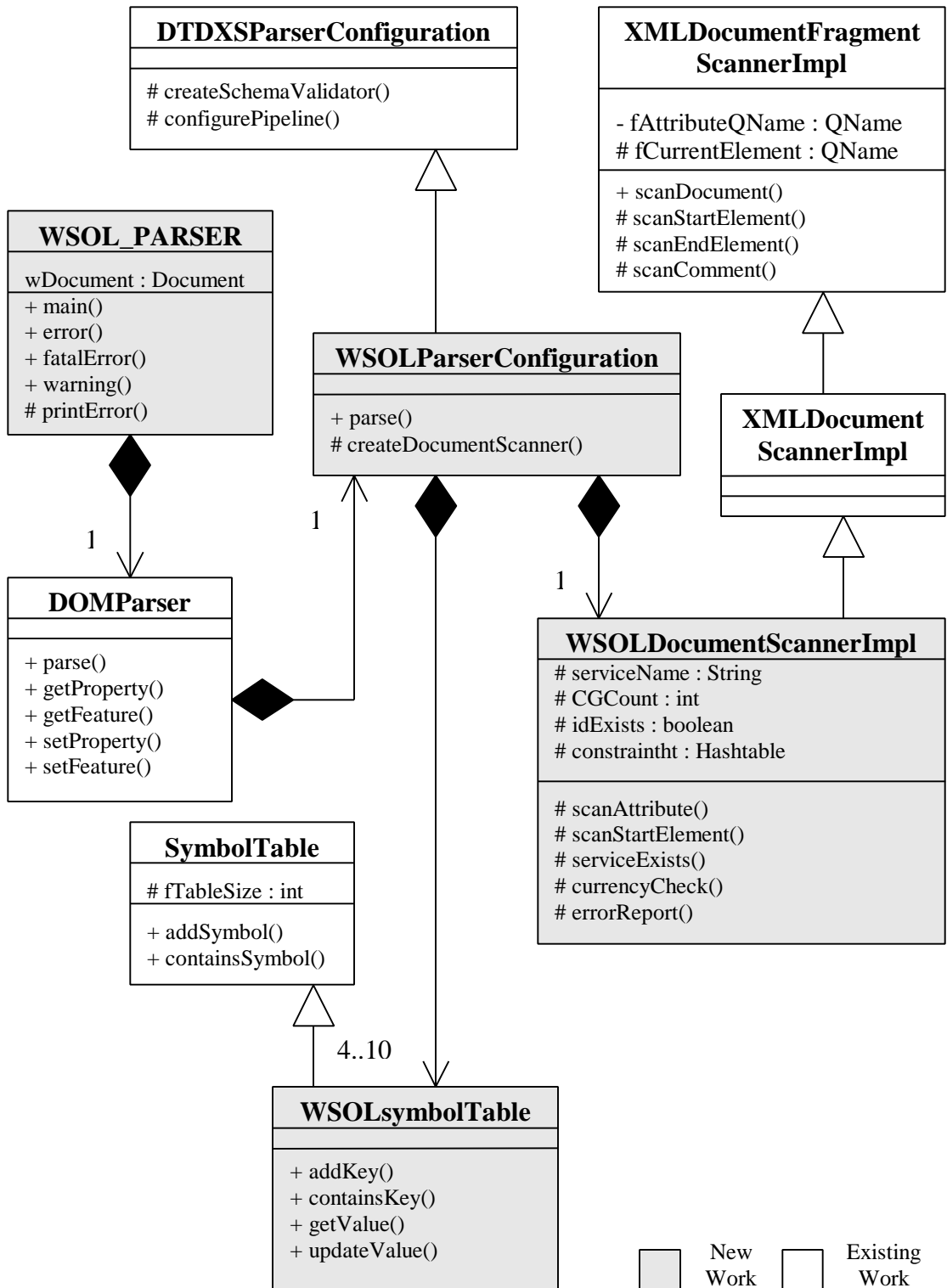
The WSOL parser, named “Premier”, developed in this Master’s thesis is a prototype parser for the WSOL language. This WSOL parser was written in the Java programming language. This chapter discusses the design details of the WSOL parser. A discussion of the features of the WSOL parser is also provided in the chapter.

This Master’s thesis focused on the development of a WSOL parser that implements WSOL language concepts and detects and reports syntax and some semantic errors. Code generation was considered outside the scope of this thesis.

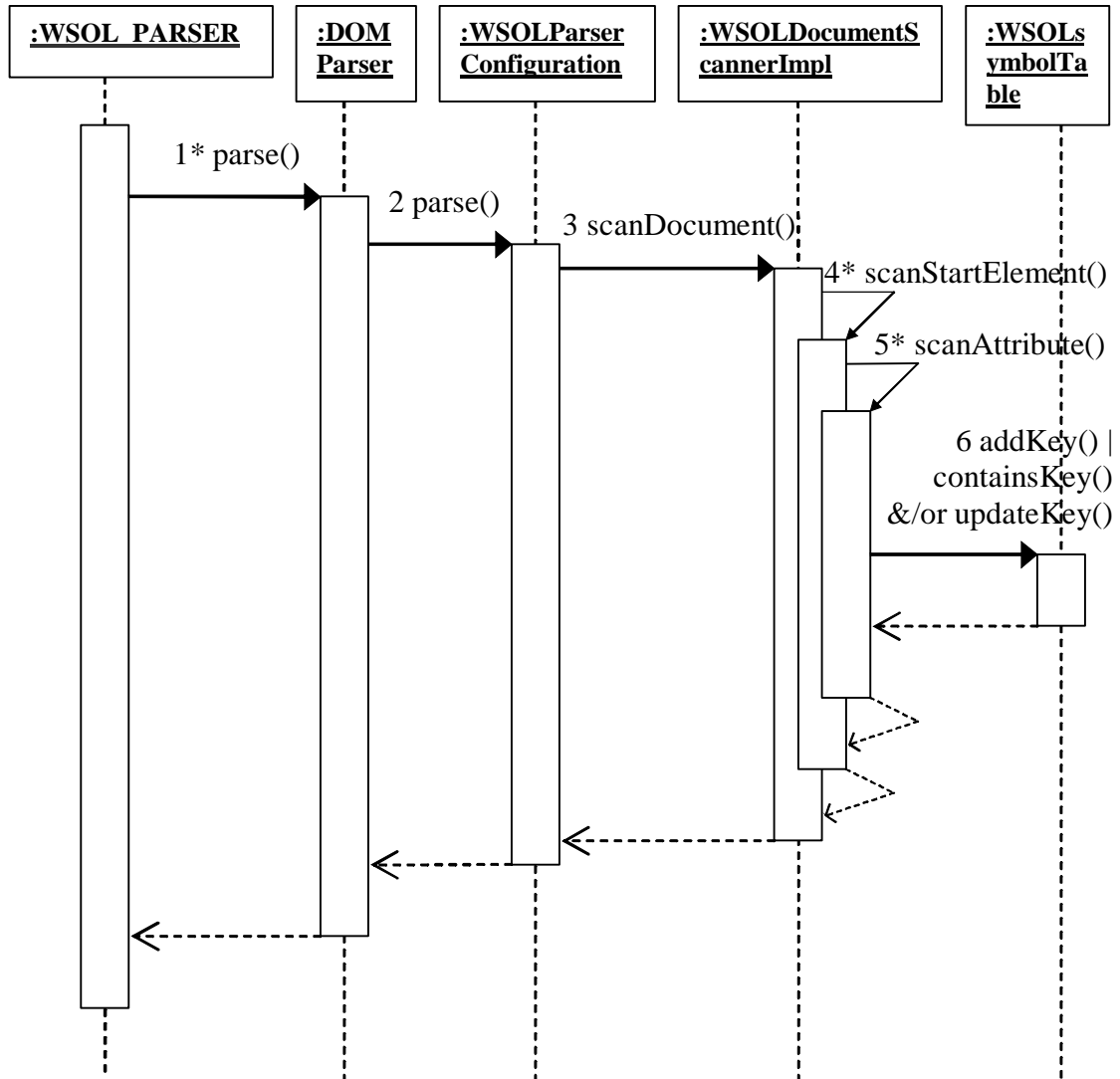
### 5.1 Design of the WSOL Parser

The WSOL parser was developed by re-using the Xerces2 Java XML parser version 2.0.1, which is an open-source XML parser developed by the Apache Software foundation. The Xerces2 Java parser and the motivation for choosing it to implement the WSOL parser have already been discussed in Section 2.4. The Xerces2 Java XML parser enables validation of an XML document against its corresponding grammar specified using either XML Schema or DTD and detection and notification of syntax errors in the XML document. It does not enable detection of semantic errors in the XML document because the grammar for the XML document contains only syntax rules for the document. The Xerces2 Java XML parser if used for validating WSOL documents would only detect syntax errors and not WSOL-related semantic errors. So, there was a need to extend the Xerces2 XML parser to detect and report semantic errors in WSOL documents. In this thesis, I have designed a WSOL parser by extending the Xerces2 Java

XML parser and adding code for detection and notification of semantic errors in WSOL documents. The design details of the WSOL parser are shown in the following UML Class diagram and Sequence diagram:



**Figure 5.1 Class Diagram of the WSOL Parser**



**Figure 5.2 Sequence Diagram of the WSOL Parser**

The WSOL parser developed in this Master’s thesis is a single pass parser. The WSOL Parser package consists of several classes, as shown in Figure 5.1. The sequence diagram in

Figure 5.2 shows interactions between the objects of classes of the WSOL parser.

Let us now look at the main features of each class of the WSOL parser:

- **WSOL\_PARSER**

- This is the main class in the WSOL parser package.
- This class receives the names of the WSOL and WSOL-related documents (such as WSDL documents and Ontology documents) to be parsed as input from the user.
- For each document to be parsed, a new WSOL\_PARSER object is created. This object contains an object of the DOMParser class. The WSOL\_PARSER object initiates parsing of a document by invoking the parse() function on the DOMParser object.
- After each document is parsed, the WSOL\_PARSER object stores and displays the parsed document's DOM tree.

- **DOMParser**

- The DOMParser class exists in the Xerces2 Java parser package.
- It generates a DOM tree of the document it parses.
- It uses a parser configuration (already available in the Xerces2 Java parser package or a new one) to parse a document.
- The DOMParser class object defined in the WSOL\_PARSER class uses the WSOL parser configuration (discussed next) to parse each document.
- The WSOL parser uses the DOM parser as its underlying parser because of the following reasons:
  - A DOM parser generates a DOM tree that contains all the elements and the attributes of a document organized in a tree structure.

- .. The generated DOM tree can be traversed and the required data can be obtained easily.
- .. The generated DOM tree can also be re-used to generate code for checking WSOL constraints and for developing a WSOL compiler.

- **WSOLParserConfiguration**

- This is a new parser configuration developed to enable the WSOL parser to detect WSOL-related syntax and semantic errors.
- It extends the DTDXS parser configuration (discussed next) already defined in the Xerces2 Java parser package.
- It creates multiple symbol table components of type (class) WSOLsymbolTable (discussed later).
- It also changes the implementation class of the parser component “scanner” from XMLDocumentScannerImpl to WSOLDocumentScannerImpl (discussed next).
- This class initiates the parsing of a document by invoking the scanDocument method on the document scanner.

- **DTDXSParserConfiguration**

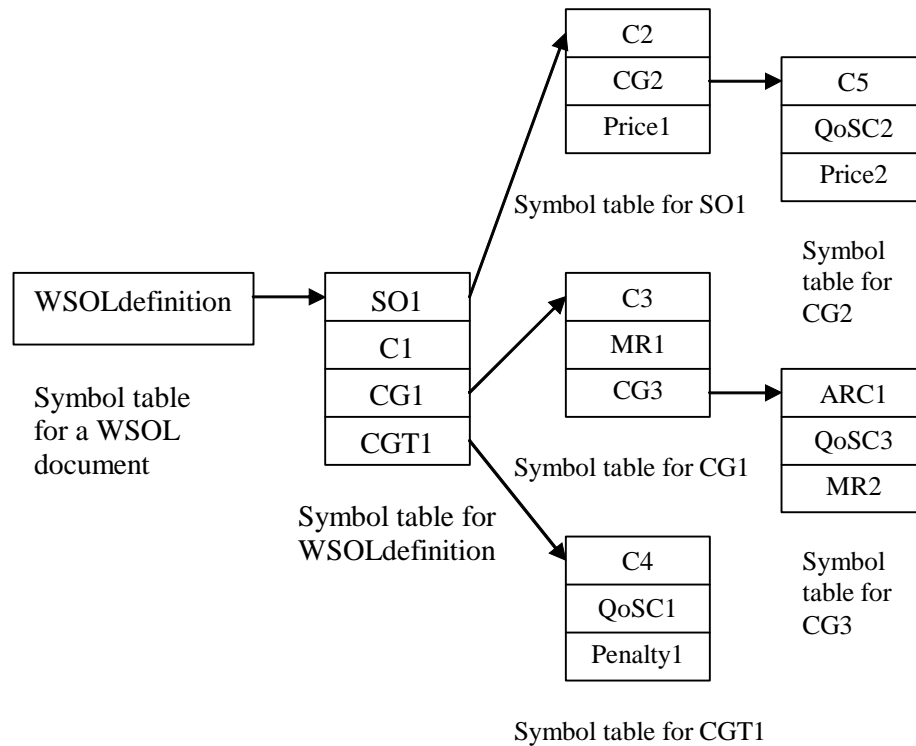
- This parser configuration extends the standard parser configuration. Both this parser configuration and the standard parser configuration are already defined in the Xerces2 Java parser package.
- The standard parser configuration contains all the standard parser components defined in the Xerces2 package, including: scanner, entity manager, and schema validator. The standard parser configuration connects all these parser components, except the schema validator component, to form a parsing pipeline.



- The DTDXS parser configuration adds the schema validator component to the end of the standard parser configuration's parsing pipeline. This enables the WSOL parser to parse the WSOL and WSOL-related documents that conform to their corresponding grammars written in the XML Schema language.
- **WSOLDocumentScannerImpl**
  - This new implementation class provides implementation for the parser component "scanner".
  - It extends an existing class XMLDocumentScannerImpl (discussed later) that also implements the scanner component.
  - The WSOLDocumentScannerImpl class enables the WSOL parser to detect and report WSOL-related syntax and semantic errors in a document.
  - It implements a function called scanAttribute that scans an attribute, detects and reports syntax and semantic errors, and stores the value of the attribute in a symbol table if needed for future reference and future semantic checks.
- **XMLDocumentScannerImpl**
  - This class is defined in the Xerces2 Java parser package.
  - It extends the XMLDocumentFragmentScannerImpl class which also exists in the Xerces2 Java parser package.
  - The class provides implementation for the parser component "scanner".
  - The class defines several functions some of which are invoked by the scanDocument function of the XMLDocumentFragmentScannerImpl class while scanning a document.

- **XMLDocumentFragmentScannerImpl**
  - This class also provides implementation for the parser component “scanner”.
  - It defines and provides implementations for functions such as, scanDocument (enables scanning a document), scanStartElement (enables scanning of the start of an element), etc.
  - The document scanner of the WSOL parser invokes the scanDocument method for scanning a document. The scanDocument function scans all elements and attributes in a document. While scanning, the scanDocument function (and other functions that it invokes) detects and reports syntax errors and semantic errors.
  - The scanAttribute function defined in this class is overridden by the scanAttribute function of the WSOLDocumentScannerImpl class.
  
- **WSOLsymbolTable**
  - This new class provides implementation for the “symbol table” parser component of the WSOL parser.
  - It extends the existing SymbolTable class and also re-uses some code from the symbolHash class in the Xerces2 Java parser package.
  - This implementation of a symbol table enables adding symbols in the form of [key, value] (where key is the symbol and the value represents some information about the symbol). It also enables retrieving information about symbols (i.e., retrieving the value of a particular key). It also allows verifying whether a particular symbol (i.e., key) exists in the symbol table. Finally, it also enables updating information about a particular symbol (i.e. updating the value of a key).

- This class implements the WSOL parser's symbol table component using hash tables as the data structure due to several reasons:
  - .. The hash tables are fast and efficient in adding data as well as retrieving data.
  - .. The hash tables enable storing the data in the form [key, value].
- When the WSOL parser stores data in the form of [key, value], the key is a simple string e.g. "SO1" (represents the name of a service offering), and the value is a hash table that contains information about the key (here "SO1"), e.g., [{service, buyStock:buyStockService}, {accountingParty, WSOL-CONSUMERWS}] represents the information for the service offering "SO1".
- A symbol table in the WSOL parser is scoped-by-structure i.e., it consists of multiple nested symbol tables created for each scope block of the document being parsed. This helps in storing and retrieving the data according to the structure of the document. If there are identifiers in a document with the same name, then a correct identifier required can be found and retrieved because of scoping. For example, consider a symbol table for a WSOL document. This symbol table when scoped by structure would look like the following:



**Figure 5.3 Example of scoping in symbol tables**

- **SymbolTable**

- This class exists in the Xerces2 Java parser package.
- It provides implementation for the symbol table parser component.
- It enables adding symbols i.e., strings to the symbol table and also verifying whether a particular symbol exists in the symbol table.
- The WSOL parser does not use this implementation of a symbol table because of the following reasons:
  - The SymbolTable class does not enable adding symbols in the form of [key, value]. For instance, if the key is a variable then the value would contain the type of the variable.

- The SymbolTable class also does not enable retrieving a symbol from the symbol table.

## 5.2 Main Features of the WSOL Parser

- Implementation of the WSOL language concepts.

- **inheritance**

The WSOL parser implements single inheritance of CGs, CGTs, and SOs (service offerings). For example, a CG named “CG1” can contain constraints, statements, and/or CGs. A new CG named “CG2” defined as an extension of “CG1” inherits all the constraints, statements, and/or CGs defined in “CG1”.

- **the inclusion mechanism**

The WSOL parser implements the inclusion mechanism in WSOL. It currently supports the inclusion of named constraints, price/penalty statements, management responsibility statements, and CGs inside a CG, CGT, SO, or outside service offerings. It does not support the inclusion of external operation call statements. Since the WSOL parser does not currently implement checks for specialization of domains, it expects the new domain of an included construct to be the same as the old domain.

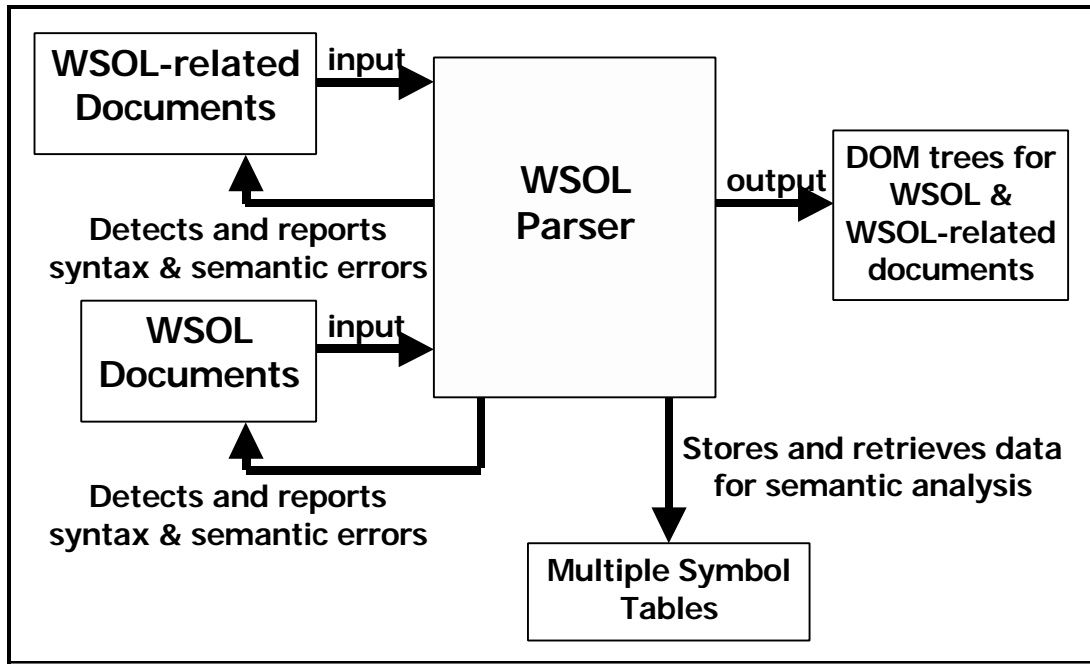
- **the CGT instantiation mechanism**

The WSOL parser implements the CGT instantiation mechanism in WSOL. It implements instantiation of a CGT inside a CG, another CGT, SO, or outside service offerings. Since the WSOL parser does not currently implement checks

for specialization of domains, it expects the new domain of an instantiated CGT to be the same as the old domain.

- Detection of syntax and some semantic errors.

The WSOL parser gets as input the WSOL-related documents and the WSOL documents to be parsed. The WSOL parser scans each document, detects and reports syntax and semantic errors in them, and creates a DOM tree as output. While scanning, the WSOL parser also stores important data (such as element names, attribute names, attribute values, attribute types, etc.) from the document being scanned into appropriate symbol tables. This stored data is used by the WSOL parser for future syntax and semantic analysis, and for verification of type compatibility. For detecting syntax and semantic errors, the WSOL parser refers to appropriate symbol tables, retrieves required data from them and compares this data with the data given in the document. An error is reported if the data do not match. Creation of multiple nested symbol tables, scoped according to the structure of the document being parsed, enables the WSOL parser to detect semantic errors. Figure 5.4 provides an overview of how a WSOL parser works.



**Figure 5.4 Overview of how a WSOL Parser works**

The WSOL parser discovers many errors. Since it is not possible to list all the errors here, only a few examples of errors detected in different WSOL constructs have been listed below:

- **Errors in Constraints**

- The referred WSDL message part specified is wrong. For example, if
  - “buyStock:buySingleStockRequest.quanty” (wrong part name),
  - “buyStock:buySingleRequest.quantity” (wrong message name),
  - “buy:buySingleStockRequest.quantity” (wrong schema name),
  - “buyStockbuySingleStockRequest.quantity” (colon “:” missing), or “ “ is specified instead of “buyStock:buySingleStockRequest.quantity”.
- The referred WSDL message part does not exist within the domain for which the constraint is specified.

- The arithmetic, string, and/or time comparisons are not type compatible. For example, if in an arithmetic comparison, a variable of data type string “name” is compared with an arithmetic constant “0” – name > 0.
- The name of an array element on which a condition is specified in a <quantified> expression is wrong.
- **Errors in Subscription Statements**
  - The unit of the subscription value is a wrong currency unit. For example, the unit of the subscription value is specified as “milliDoll” instead of “milliDollar”.
- **Errors in Price Default and Penalty Default Statements**
  - The default price is a negative value or the default penalty is a positive value.
- **Errors in Price and Penalty Statements**
  - The domain for which the price or penalty statement is specified does not exist.
- **Errors in Management Responsibility Statements**
  - The scope of management responsibility is not a constraint, CG, or a service offering
  - The value of the scope of management responsibility is wrong.
- **Errors in External Operation Call Statements**
  - The description of the called external operation is wrong. The description of the called external operation consists of the name of the external operation, the name of the port type to which the external operation belongs, and the name of the namespace in which both the external operation and the port type



are defined, for example, “symbolLookup:symbolLookupPortType.symbolExists”. The description of the called external operation is wrong if the name of the external operation is wrong “symbolLookup:symbolLookupPortType.Exists”, if the name of the port type is wrong “symbolLookup:PortType.symbolExists”, or if the name of the namespace is wrong “Lookup:symbolLookupPortType.symbolExists”.

- **Errors in Include Statements**

- The name of the WSOL construct to be included is wrong.

- **Errors in CGT Instantiation Statements**

- The name of the CGT parameter for which a value is provided is wrong.

- **Errors in CGs**

- The value of the “*extends*” attribute is “WSOL-NONE”.
- The name of an attribute specified in a CG is wrong. For example, instead of “*portOrPortType*” the name of the attribute is specified as “*portOr*”.

- **Errors in CGTs**

- The attribute “*unit*” of the CGT parameter is specified even if the data type is not “numberWithUnit”.
- The value of the “*unit*” attribute of the CGT parameter is “WSOL-NOUNIT”.

- **Errors in Service Offerings (SOs)**

- The value of the “*accountingParty*” attribute is “WSOL-SUPPLIERWS”.
- The rules for the constants “WSOL-ANY”, “WSOL-MANY”, “WSOL-ALL” and “WSOL-EVERY” used as values of domain attributes are not obeyed.

- The domain attributes “*portOrPortType*” and “*operation*” are mentioned for a service offering construct.
- Notification of all detected errors

The WSOL parser reports all the detected syntax and semantic errors to the user of the tool. The WSOL parser notifies the user about the location of the error (i.e., file name and line number) and provides a brief description of the error.

### **5.3 Summary**

The design details of the developed WSOL parser were discussed in this chapter. Also, the main features of the WSOL parser were listed in this chapter. The main features are implementation of WSOL language concepts and detection and notification of syntax and semantic errors.

## 6 Demonstration of the WSOL Parser

---

This chapter demonstrates the ability of the WSOL parser to recognize WSOL statements and to detect and report syntax and semantic errors in them. The buyStock Web Service example, discussed in Section 2.2.2, is used in this chapter for the purpose of demonstration.

For the buyStock Web Service, the WSDL document “buyStockService.wsdl” and the WSOL document “serviceOfferings.wsol” were created. These documents are given in Appendix D and Appendix E respectively. Three additional Web Services (mathLibrary Web Service, symbolLookup Web Service, and time Web Service) were also created. For the mathLibrary Web Service, the WSDL document “mathLibraryService.wsdl” was created. For the symbolLookup Web Service, the WSDL document “symbolLookupService.wsdl” was created. For the time Web Service, the WSDL document “timeService.wsdl” was created. The mathLibrary Web Service currently provides two math operations – a) calculating the sum of array elements of type “float” and b) calculating the sum of array elements of type “integer”. The symbolLookup Web Service currently provides one operation - verifying whether a stock symbol of a company exists or not. The time Web Service currently provides two operations – a) getting the current time and b) getting the current date time. The operations provided by these three Web Services are called by the constraint-checking code while checking the constraints specified for the operations of the buyStock Web Service to demonstrate the external operation call mechanism in WSOL.

Three example ontology files “QoSMetricOntology.ont”, “QoSMeasOntology.ont”, and “CurrencyOntology.ont” were also created. The file “QoSMeasOntology.ont” is given in Appendix F to show an example ontology file. The files “QoSMetricOntology.ont”, “QoSMeasOntology.ont”, and “CurrencyOntology.ont” currently consist of only a simple collection of names of QoS metrics, measurement units, and currency units respectively. These metrics and units are referred in the WSOL document “serviceOfferings.wsol” specified for the buyStock Web Service. Web Service suppliers should use standard ontologies of QoS metrics, measurement units, and currency units but if they want they can also define their own QoS metrics, measurement units, and currency units that are not there in standard ontologies.

For the demonstration purposes, the WSOL parser was made to parse all the above mentioned WSDL documents, followed by all the Ontology documents, and finally the WSOL document created for the buyStock Web Service. Initially, no errors were introduced in the WSOL document and the ability of the WSOL parser to parse a document with correct WSOL grammar was checked. It was seen that the WSOL parser was able to successfully parse the WSOL document. In this chapter, a small portion of the WSOL document that is parsed correctly by the WSOL parser is shown along with the output of the WSOL parser and the information regarding the symbol tables created. Later on, many errors were deliberately introduced in the WSOL document so that the parser could detect and report them. The WSOL parser was able to detect and report all the errors introduced. In this chapter, only a few examples of the errors that were introduced, detected, and reported by the WSOL parser are shown. A screen shot is also

shown for each example. These screen shots demonstrate that the WSOL parser is able to detect and report the introduced errors.

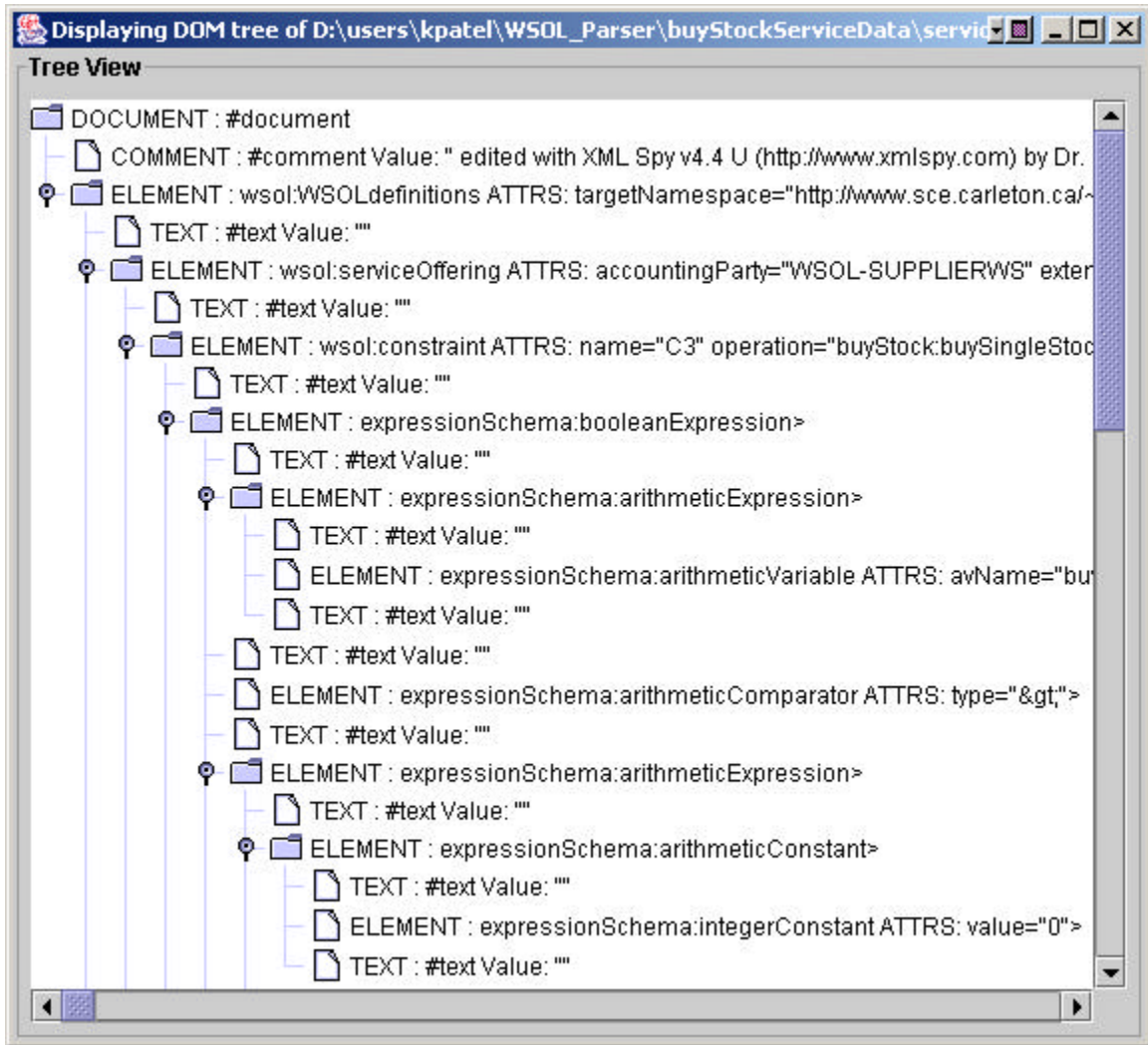
### 6.1 An example of correct parsing

A syntactically and semantically correct segment of the WSOL document (“serviceOfferings.wsol”) is shown below:

```
<wsol:serviceOffering name = "SO1" service = "buyStock:buyStockService">
  <wsol:constraint name = "C3" xsi:type = "preConditionSchema:preCondition"
  service = "buyStock:buyStockService" portOrPortType =
  "buyStock:buyStockServicePort" operation =
  "buyStock:buySingleStockOperation">
    <expressionSchema:booleanExpression>
      <expressionSchema:arithmeticExpression>
        <expressionSchema:arithmeticVariable avName =
        "buyStock:buySingleStockRequest.quantity"/>
      </expressionSchema:arithmeticExpression>
      <expressionSchema:arithmeticComparator type = ">"/>
      <expressionSchema:arithmeticExpression>
        <expressionSchema:arithmeticConstant>
          <expressionSchema:integerConstant value =
          "0"/>
        </expressionSchema:arithmeticConstant>
      </expressionSchema:arithmeticExpression>
    </expressionSchema:booleanExpression>
  </wsol:constraint>
</wsol:serviceOffering>
```

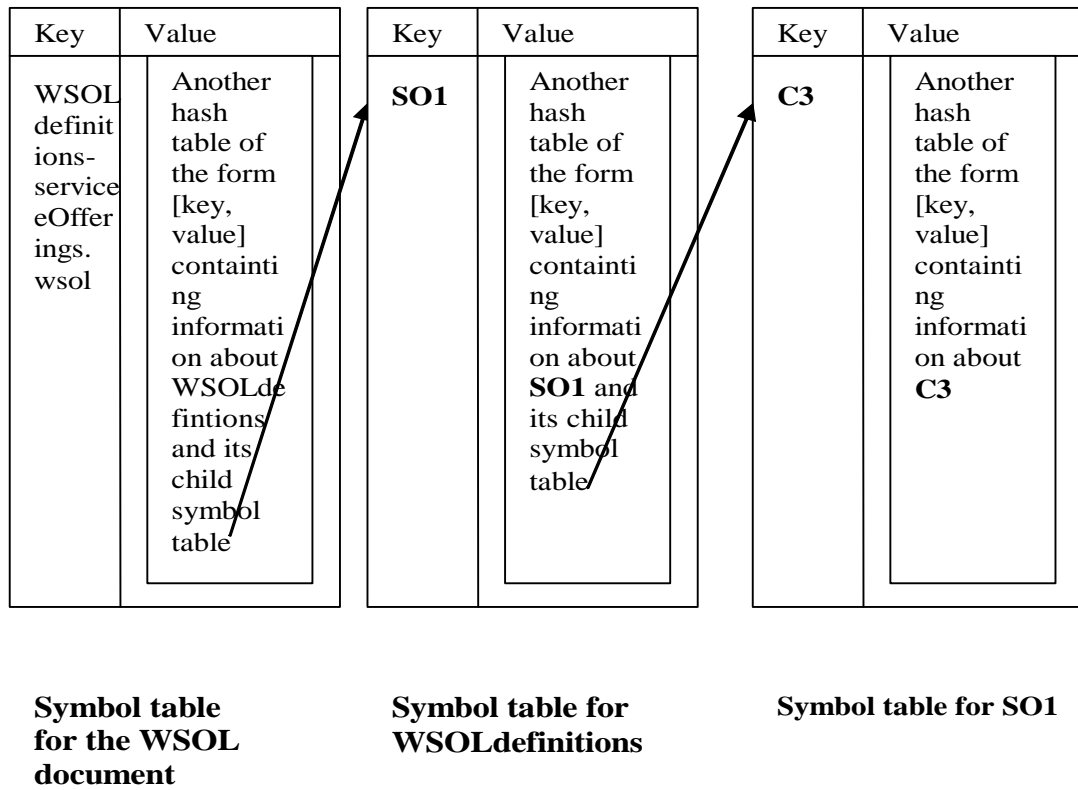
**Figure 6.1** A syntactically and semantically correct segment of the WSOL document

The WSOL parser parses this segment successfully and generates a DOM tree as output as shown below:



**Figure 6.2 DOM tree of the parsed WSOL document**

The WSOL parser while parsing the WSOL segment also stores important information from the segment into nested symbol tables as shown below:



**Figure 6.3 Symbol Tables generated for the parsed WSOL document**

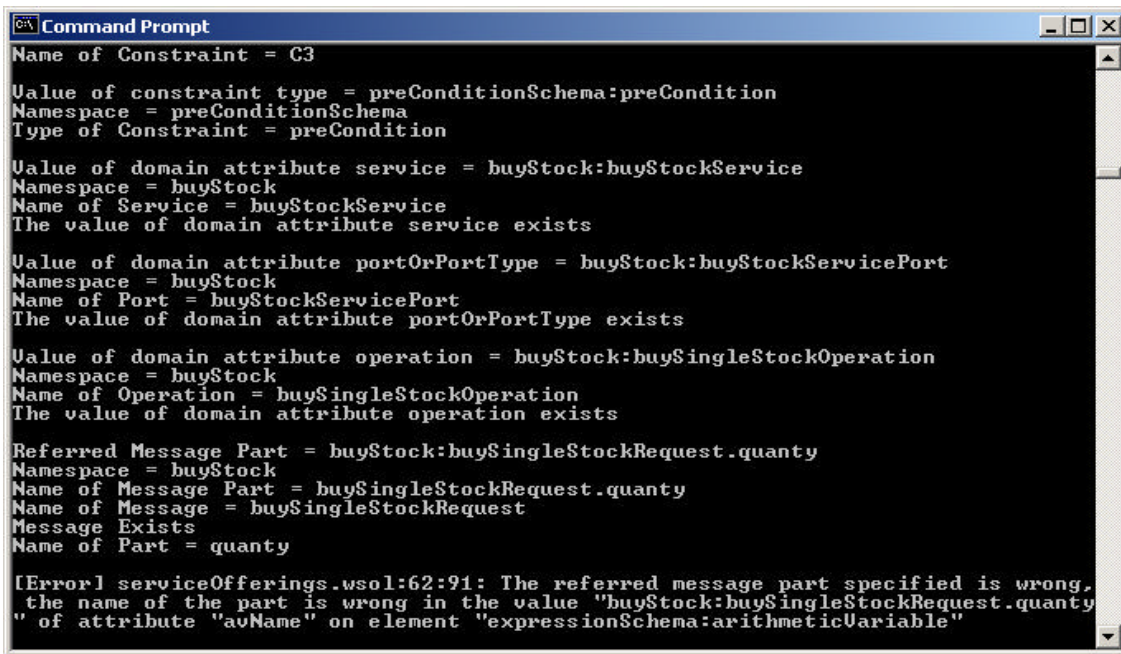
This data stored in the symbol tables is used by the WSOL parser for future syntax and semantic analysis, and for verification of type compatibility. The WSOL parser will be extended in future to use the data stored in the symbol tables and the generated DOM tree for generating constraint-checking code in Java for checking whether the constraints specified in a WSOL document are satisfied or not.

## 6.2 Examples of discovered syntax and semantic errors

The errors shown below are introduced in the WSOL constructs—i.e., in constraints, statements, CGs, CGTs, and service offerings—specified in the “serviceOfferings.wsol” document.

### 6.2.1 Errors in Constraints

1. The “serviceOfferings.wsol” document contains a functional constraint named “C3”. The constraint “C3” refers to the message part “buyStock:buySingleStockRequest.quantity”. Now, in this referred message part, if the name of the part (i.e., **quantity**) is changed to “**quandy**”, then the referred message part becomes “buyStock:buySingleStockRequest.**quandy**”. This new referred message part is wrong, because the name of the part is wrong. The WSOL parser detects and reports this error as shown in the following screen shot:



```
Command Prompt
Name of Constraint = C3
Value of constraint type = preConditionSchema:preCondition
Namespace = preConditionSchema
Type of Constraint = preCondition
Value of domain attribute service = buyStock:buyStockService
Namespace = buyStock
Name of Service = buyStockService
The value of domain attribute service exists
Value of domain attribute portOrPortType = buyStock:buyStockServicePort
Namespace = buyStock
Name of Port = buyStockServicePort
The value of domain attribute portOrPortType exists
Value of domain attribute operation = buyStock:buySingleStockOperation
Namespace = buyStock
Name of Operation = buySingleStockOperation
The value of domain attribute operation exists
Referred Message Part = buyStock:buySingleStockRequest.quandy
Namespace = buyStock
Name of Message Part = buySingleStockRequest.quandy
Name of Message = buySingleStockRequest
Message Exists
Name of Part = quandy
[Error] serviceOfferings.wsol:62:91: The referred message part specified is wrong,
the name of the part is wrong in the value "buyStock:buySingleStockRequest.quandy
" of attribute "avName" on element "expressionSchema:arithmeticVariable"
```

Figure 6.4 An error detected in a constraint



- The next error is also introduced in the constraint “C3” mentioned earlier. The constraint “C3” is specified for a particular domain (i.e., for *service* = “buyStock:buyStockService”, *portOrPortType* = “buyStock:buyStockServicePort”, and *operation* = “buyStock:buySingleStockOperation”).

This domain exists and also the referred message part “buyStock:buySingleStockRequest.quantity” specified within the constraint “C3” exists within this domain. Now, if the value of the domain attribute *operation* is changed to “buyStock:buySingleOperation”, the new domain for which the constraint “C3” is specified does not exist since the operation name is wrong. Also, the referred message part specified within the constraint “C3” does not exist within the new domain. The WSOL parser detects and reports these errors as shown in the following screen shot:

```

Command Prompt
Value of domain attribute operation = buyStock:buySingleOperation
Namespace = buyStock
Name of Operation = buySingleOperation
The value of domain attribute operation does not exist

[Error] serviceOfferings.wsol:59:202: The domain does not exist, the name of operation is wrong in the value "buyStock:buySingleOperation" of attribute "operation" on element "wsol:constraint"

Referred Message Part = buyStock:buySingleStockRequest.quantity
Namespace = buyStock
Name of Message Part = buySingleStockRequest.quantity
Name of Message = buySingleStockRequest
Message Exists

[Error] serviceOfferings.wsol:62:93: The referred message part does not exist with in the domain for which the constraint is specified in the value "buyStock:buySingleStockRequest.quantity" of attribute "avName" on element "expressionSchema:arithmeticVariable"

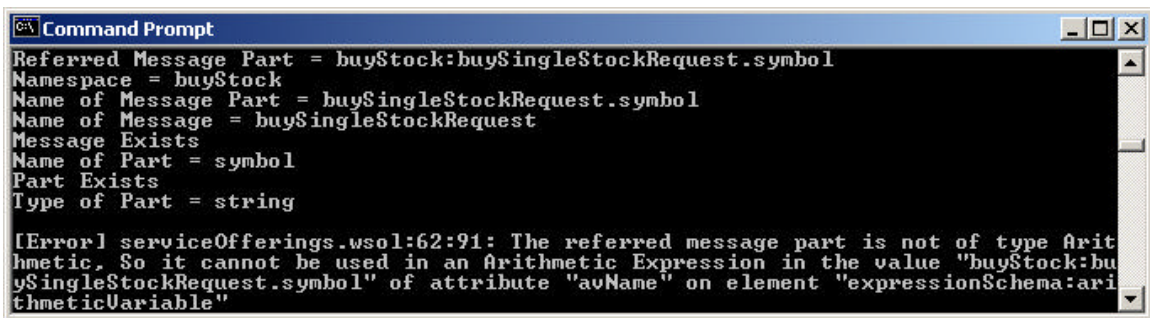
Name of Part = quantity
Part Exists

```

**Figure 6.5 An error detected in a constraint**

- The next error is also introduced in the constraint “C3” mentioned earlier. The constraint “C3” contains a comparison of two arithmetic expressions. An

arithmetic variable “buyStock:buySingleStockRequest.**quantity**” of data type “integer” is compared with the arithmetic constant “0”. Now, if this arithmetic variable is replaced by another variable “buyStock:buySingleStockRequest.**symbol**” of the data type “string”, then the constraint “C3” will contain a comparison of a variable of data type “string” with an arithmetic constant, which is wrong and not type compatible. The WSOL parser detects and reports this error as shown in the following screen shot:



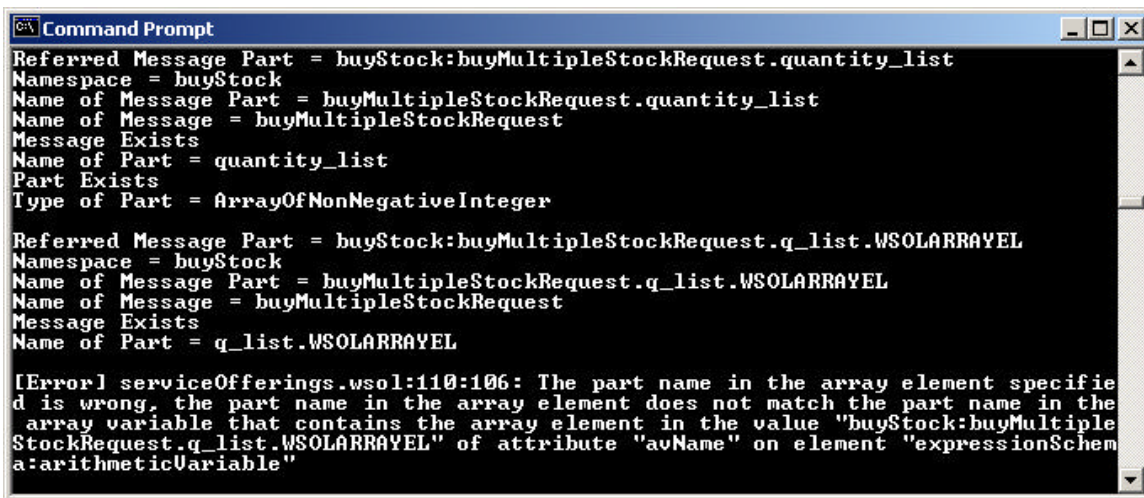
```
Command Prompt
Referred Message Part = buyStock:buySingleStockRequest.symbol
Namespace = buyStock
Name of Message Part = buySingleStockRequest.symbol
Name of Message = buySingleStockRequest
Message Exists
Name of Part = symbol
Part Exists
Type of Part = string

[Error] serviceOfferings.wsol:62:91: The referred message part is not of type Arithmetic. So it cannot be used in an Arithmetic Expression in the value "buyStock:buySingleStockRequest.symbol" of attribute "avName" on element "expressionSchema:arithmeticVariable"
```

**Figure 6.6 An error detected in a constraint**

4. Another constraint, specified in the “serviceOfferings.wsol” document, contains a quantified expression consisting of a quantifier (ForAll), an array variable “buyStock:buyMultipleStockRequest.**quantity\_list**”, and an array element “buyStock:buyMultipleStockRequest.**quantity\_list**.WSOLARRAYEL” on which some condition is specified. The array element should belong to the array specified in the array variable. In other words, the namespace, message, and part specified in the array element must match the namespace, message, and part specified in the array variable. Now, if in the array element the part “**quantity\_list**” is replaced by “**q\_list**”, then the part “**q\_list**” does not match the part “**quantity\_list**” specified in the array variable. As a result, the name of the array element is wrong and the array element does not belong to the array

specified in the array variable. The WSOL parser detects and reports this error as shown in the following screen shot:



```
Command Prompt
Referred Message Part = buyStock:buyMultipleStockRequest.quantity_list
Namespace = buyStock
Name of Message Part = buyMultipleStockRequest.quantity_list
Name of Message = buyMultipleStockRequest
Message Exists
Name of Part = quantity_list
Part Exists
Type of Part = ArrayOfNonNegativeInteger

Referred Message Part = buyStock:buyMultipleStockRequest.q_list.WSOLARRAYEL
Namespace = buyStock
Name of Message Part = buyMultipleStockRequest.q_list.WSOLARRAYEL
Name of Message = buyMultipleStockRequest
Message Exists
Name of Part = q_list.WSOLARRAYEL

[Error] serviceOfferings.wsol:110:106: The part name in the array element specified is wrong, the part name in the array element does not match the part name in the array variable that contains the array element in the value "buyStock:buyMultipleStockRequest.q_list.WSOLARRAYEL" of attribute "avName" on element "expressionSchema:arithmeticVariable"
```

Figure 6.7 An error detected in a constraint

## 6.2.2 Errors in Subscription Statements

1. A subscription statement, specified in the “serviceOfferings.wsol” document, contains the unit of subscription value specified as “milliDoll”. The correct name of the currency unit is “milliDollar”. The WSOL parser detects and reports this error as shown in the following screen shot:

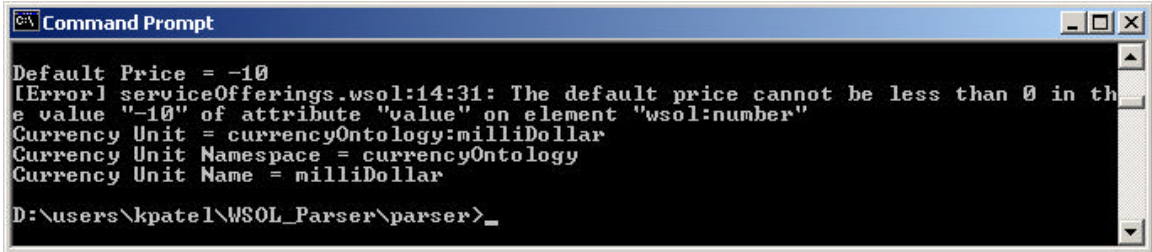


```
Command Prompt
Subscription Value = 5000
Currency Unit = currencyOntology:milliDoll
Currency Unit Namespace = currencyOntology
Currency Unit Name = milliDoll
[Error] serviceOfferings.wsol:8:50: Wrong Currency Unit Name in the value "currencyOntology:milliDoll" of attribute "type" on element "wsol:unit"
D:\users\kpatel\WSOL_Parser\parser>
```

Figure 6.8 An error detected in a subscription statement

### 6.2.3 Errors in Price Default/Penalty Default Statements

1. A price default statement, specified in the “serviceOfferings.wsol” document, contains a default price of “-10 milliDollar”. A default price cannot be less than 0. The WSOL parser detects and reports this error as shown in the following screen shot:



```
Command Prompt
Default Price = -10
[Error] serviceOfferings.wsol:14:31: The default price cannot be less than 0 in the
value "-10" of attribute "value" on element "wsol:number"
Currency Unit = currencyOntology:milliDollar
Currency Unit Namespace = currencyOntology
Currency Unit Name = milliDollar
D:\users\kpatel\WSOL_Parser\parser>
```

Figure 6.9 An error detected in a price default statement

### 6.2.4 Errors in Price/Penalty Statements

1. A price statement named “Price1” is specified in the “serviceOfferings.wsol” document for the domain *service* = “buyStock:buyStockService”, *portOrPortType* = “buyStock:buyStockServicePort”, and *operation* = “buyStock:buySingleOperation”. This domain does not exist because the name of the operation specified in the domain attribute “*operation*” is wrong. It should have been *operation* = “buyStock:buySingleStockOperation”. The WSOL parser detects and reports this error as shown in the following screen shot:

```
Command Prompt
Name of Price Statement = Price1
Value of domain attribute service = buyStock:buyStockService
Namespace = buyStock
Name of Service = buyStockService
The value of domain attribute service exists

Value of domain attribute portOrPortType = buyStock:buyStockServicePort
Namespace = buyStock
Name of Port = buyStockServicePort
The value of domain attribute portOrPortType exists

Value of domain attribute operation = buyStock:buySingleOperation
Namespace = buyStock
Name of Operation = buySingleOperation
The value of domain attribute operation does not exist

[Error] serviceOfferings.wsol:24:156: The domain does not exist, the name of operation is wrong in the value "buyStock:buySingleOperation" of attribute "operation" on element "wsol:price"

Price = 3
Currency Unit = currencyOntology:milliDollar
Currency Unit Namespace = currencyOntology
Currency Unit Name = milliDollar
D:\users\kpatel\WSOL_Parser\parser>
```

Figure 6.10 An error detected in a price statement

### 6.2.5 Errors in Management Responsibility Statements

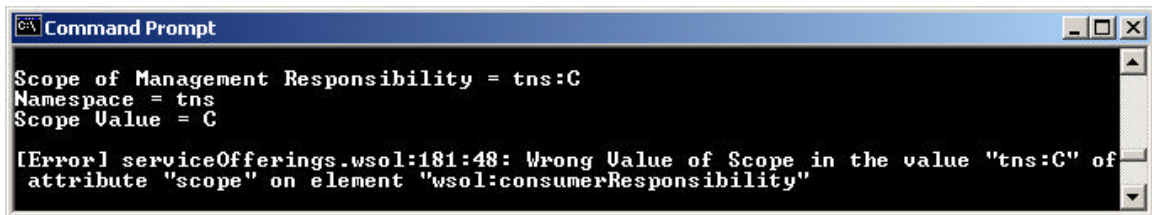
1. A management responsibility statement is specified in the WSOL document “serviceOfferings.wsol” with the scope of supplier responsibility specified as a the price statement “tns:Price1”. In WSOL, the scope of management responsibility can only be a constraint, CG, or a service offering. The value of a scope cannot be a price statement. The WSOL parser detects and reports this error as shown in the following screen shot:

```
Command Prompt
Scope of Management Responsibility = tns:Price1
Namespace = tns
Scope Value = Price1

[Error] serviceOfferings.wsol:181:53: Scope of management responsibility should only be a constraint, CG, or service offering in the value "tns:Price1" of attribute "scope" on element "wsol:consumerResponsibility"
```

Figure 6.11 An error detected in a management responsibility statement

2. Another management responsibility statement is specified in the WSOL document “serviceOfferings.wsol” with the scope of supplier responsibility specified as a the constraint “tns:C3”. The WSOL parser verifies that the constraint named “C3” actually exists in the WSOL document. Now, if instead of “tns:C3”, the value of scope is specified as “tns:C”, the WSOL parser again verifies whether the constraint named “C” actually exists in the WSOL document. But, since there is no constraint named “C” defined in the WSOL document, the value of the scope is wrong. The WSOL parser detects and reports this error as shown in the following screen shot:



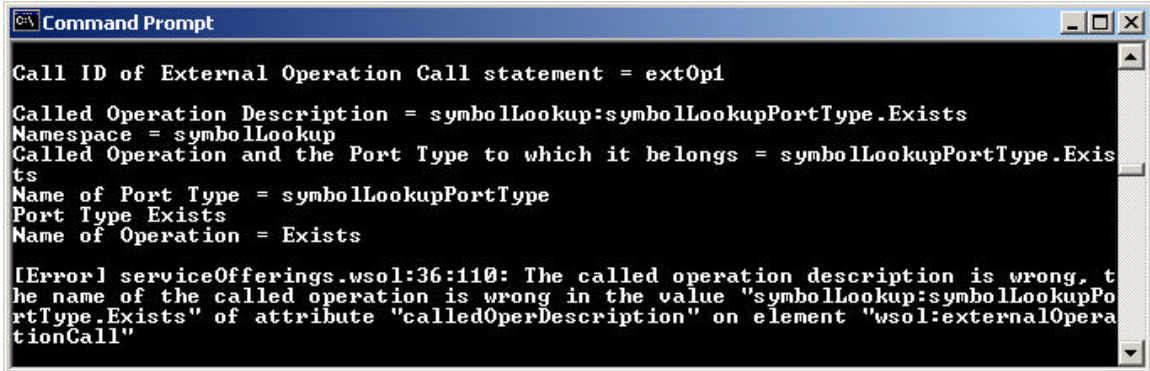
```
Command Prompt
Scope of Management Responsibility = tns:C
Namespace = tns
Scope Value = C
[Error] serviceOfferings.wsol:181:48: Wrong Value of Scope in the value "tns:C" of
attribute "scope" on element "wsol:consumerResponsibility"
```

**Figure 6.12 An error detected in a management responsibility statement**

### **6.2.6 Errors in External Operation Call Statements**

1. An external operation call statement is specified in the “serviceOfferings.wsol” document to provide information about an external operation call made during constraint evaluation to the symbolLookup Web Service. This external operation call statement provides a description of the called operation “symbolLookup:symbolLookupPortType.symbolExists” that includes the name of the called operation “symbolExists” and the name of the port type “symbolLookupPortType” to which the called operation belongs. Now, if, in the same external operation call statement, the name of the called operation

“symbolExists” is replaced by “Exists”, then the called operation description becomes “symbolLookup:symbolLookupPortType.Exists”. This called operation description is wrong since there is no operation named “Exists” defined by the symbolLookup Web Service. The WSOL parser detects and reports this error as shown in the following screen shot:



```
Command Prompt
Call ID of External Operation Call statement = extOp1
Called Operation Description = symbolLookup:symbolLookupPortType.Exists
Namespace = symbolLookup
Called Operation and the Port Type to which it belongs = symbolLookupPortType.Exists
Name of Port Type = symbolLookupPortType
Port Type Exists
Name of Operation = Exists

[Error] serviceOfferings.wsol:36:110: The called operation description is wrong, the name of the called operation is wrong in the value "symbolLookup:symbolLookupPortType.Exists" of attribute "calledOperationDescription" on element "wsol:externalOperationCall"
```

Figure 6.13 An error detected in an external operation call statement

### 6.2.7 Errors in Include Statements

1. A service offering “SO2” is specified in the “serviceOfferings.wsol” document. “SO2” includes a constraint “C3” defined in another service offering “SO1” using an Include statement. In the Include statement, if it is specified that constructName=”tns:SO1.C” instead of constructName=”tns:SO1.C3” then it is an error because the name of the WSOL construct to be included is wrong. The WSOL parser detects and reports this error as shown in the following screen shot:

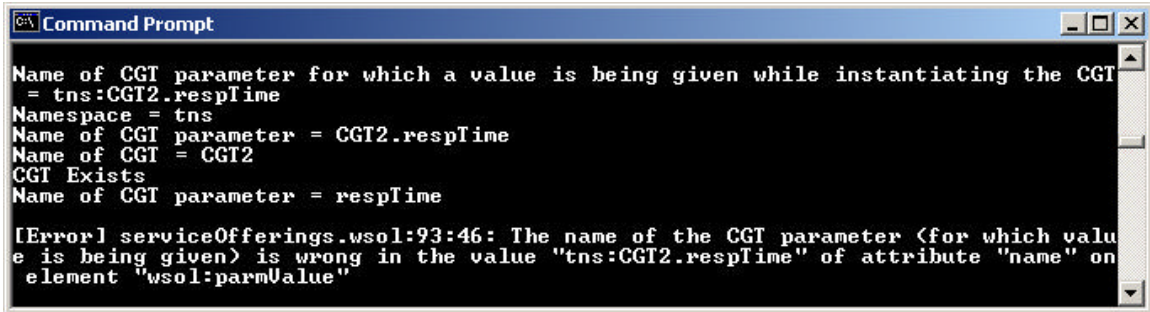
```
Command Prompt
Name of Service Offering = S02
Value of domain attribute service = buyStock:buyStockService
Namespace = buyStock
Name of Service = buyStockService
The value of domain attribute service exists
Namespace = tns
Name of Construct = S01.C
[Error]
Location of Error -> FileName(serviceOfferings.wsol) : Line(70) : Column(43) : Element(wsol:include): Attribute(constructName): AttributeValue(S01.C)
Cause of Error -> Name does not exist
D:\users\kpatel\WSOL_Parser\parser>
```

Figure 6.14 An error detected in an Include statement

### 6.2.8 Errors in CGT Instantiation Statements

1. A CGT instantiation statement specified in the “serviceOfferings.wsol” document instantiates the CGT named “CGT2” (i.e., this statement provides values for the parameters defined in the CGT “CGT2”). One of the parameters defined by the CGT “CGT2” is called “responseTime”. In this CGT instantiation statement this CGT parameter is referred to as “tns:CGT2.responseTime” and a value is provided for this parameter. Now, if in the referred CGT parameter “tns:CGT2.responseTime”, the parameter name “responseTime” is changed to “respTime”, then the referred CGT parameter becomes “tns:CGT2.respTime”. This referred CGT parameter is wrong since the parameter name “respTime” does not exist and does not match the parameter name “responseTime” defined in the CGT “CGT2”. The WSOL parser detects and reports this error as shown in the following screen shot:





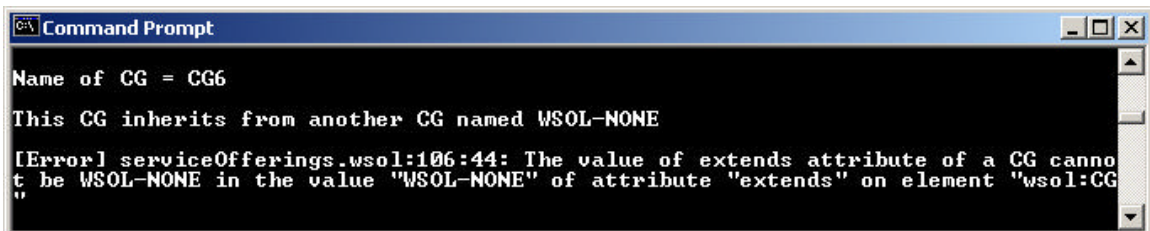
```
Command Prompt
Name of CGI parameter for which a value is being given while instantiating the CGI
= tns:CGT2.respTime
Namespace = tns
Name of CGI parameter = CGI2.respTime
Name of CGI = CGI2
CGI Exists
Name of CGI parameter = respTime

[Error] serviceOfferings.wsol:93:46: The name of the CGI parameter (for which value
is being given) is wrong in the value "tns:CGT2.respTime" of attribute "name" on
element "wsol:parmValue"
```

Figure 6.15 An error detected in a CGI Instantiation statement

### 6.2.9 Errors in CGs

1. A CG named “CG6” is specified in the “serviceOfferings.wsol” document. In WSOL, a CG can inherit from another CG that already exists. The inheritance information is provided with the attribute “*extends*”. For the CG “CG6”, the value for the “*extends*” attribute is specified as “WSOL-NONE”. The value of the “*extends*” attribute for a CG can only be the name of a particular CG. It cannot be “WSOL\_NONE”. The “WSOL-NONE” constant is used in WSOL as a default value if the attribute “*extends*” is not specified for a CG. The WSOL parser detects and reports this error as shown in the following screen shot:



```
Command Prompt
Name of CG = CG6
This CG inherits from another CG named WSOL-NONE

[Error] serviceOfferings.wsol:106:44: The value of extends attribute of a CG cannot
be WSOL-NONE in the value "WSOL-NONE" of attribute "extends" on element "wsol:CG"
```

Figure 6.16 An error detected in a CG

2. A CG named “CG6” specified in the “serviceOfferings.wsol” document has an attribute `service="buyStock:buyStockService"` instead of `service="`

buyStock:buyStockService”. The WSOL parser detects and reports this syntax error as shown in the following screen shot:

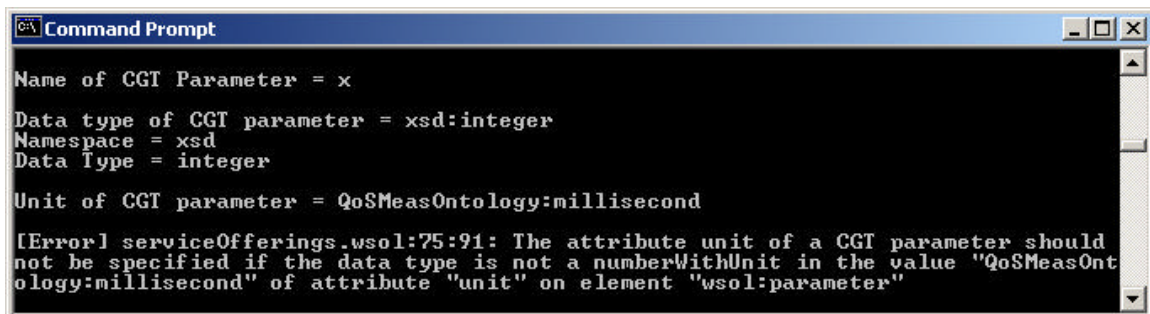


```
Command Prompt
Name of CG = CG6
[Error]
Location of Error -> FileName(serviceOfferings.wsol) : Line(112) : Column(155) :
cvc-complex-type.4: Attribute 'service' must appear on element 'wsol:CG'.
[Error]
Location of Error -> FileName(serviceOfferings.wsol) : Line(112) : Column(155) :
cvc-complex-type.3.2.2: Attribute 'service' is not allowed to appear in element
'wsol:CG'.
```

Figure 6.17 An error detected in a CG

### 6.2.10 Errors in CGTs

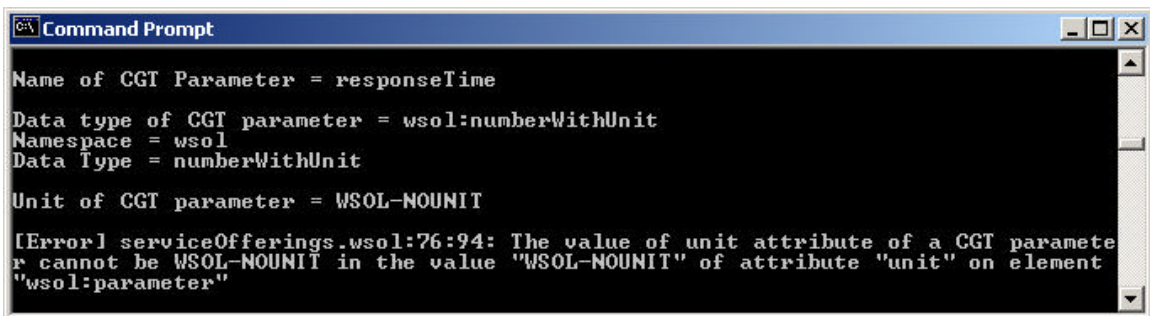
1. A CGT parameter named “x” is specified within a CGT in the “serviceOfferings.wsol” document. The data type of the CGT parameter is specified as “integer” and the unit of CGT parameter is specified as “QoSMeasOntology:millisecond”. According to the WSOL language, the specification of attribute “unit” of a CGT parameter is optional and it should be specified only if the data type of a CGT parameter is “numberWithUnit”. In the CGT parameter “x”, the attribute “unit” is specified even if the data type is “integer”. The WSOL parser detects and reports this error as shown in the following screen shot:



```
Command Prompt
Name of CGT Parameter = x
Data type of CGT parameter = xsd:integer
Namespace = xsd
Data type = integer
Unit of CGT parameter = QoSMeasOntology:millisecond
[Error] serviceOfferings.wsol:75:91: The attribute unit of a CGT parameter should
not be specified if the data type is not a numberWithUnit in the value "QoSMeasOnt
ology:millisecond" of attribute "unit" on element "wsol:parameter"
```

Figure 6.18 An error detected in a CGT

2. A CGT parameter named “responseTime” is specified within a CGT in the “serviceOfferings.wsol” document. The unit of the CGT parameter is specified as “WSOL-NOUNIT”. According to the WSOL language, if the attribute “unit” is specified for a CGT parameter, then its value should not be “WSOL-NOUNIT”, but a particular unit. The “WSOL-NOUNIT” constant is used in WSOL as the default value if the attribute “unit” is not specified for a CGT parameter. The WSOL parser detects and reports this error as shown in the following screen shot:



```
Command Prompt
Name of CGT Parameter = responseTime
Data type of CGT parameter = wsol:numberWithUnit
Namespace = wsol
Data Type = numberWithUnit
Unit of CGT parameter = WSOL-NOUNIT
[Error] serviceOfferings.wsol:76:94: The value of unit attribute of a CGT parameter cannot be WSOL-NOUNIT in the value "WSOL-NOUNIT" of attribute "unit" on element "wsol:parameter"
```

**Figure 6.19 An error detected in a CGT**

### **6.2.11 Errors in Service Offerings (SOs)**

1. A service offering named “SO1” is specified in the “serviceOfferings.wsol” document. The value of attribute “accountingParty” for “SO1” is specified as “WSOL-SUPPLIERWS”. According to the WSOL language, if the attribute “accountingParty” is specified explicitly for a service offering, then its value should not be “WSOL-SUPPLIERWS”. The “WSOL-SUPPLIERWS” constant is used in WSOL as the default value if the attribute “accountingParty” is not specified for a service offering. The WSOL parser detects and reports this error as shown in the following screen shot:

```

Command Prompt
Name of Service Offering = S01
Value of domain attribute service = buyStock:buyStockService
Namespace = buyStock
Name of Service = buyStockService
The value of domain attribute service exists
Accounting Party for Service Offering = WSOL-SUPPLIERWS
[Error] serviceOfferings.wsol:59:108: WSOL-SUPPLIERWS is the default value, it can
not be specified as the value of attribute accountingParty in the value "WSOL-SUPP
LIERWS" of attribute "accountingParty" on element "wsol:serviceOffering"

```

**Figure 6.20 An error detected in a service offering**

2. This error is also introduced in the service offering “SO1” mentioned earlier. If the service offering “SO1” is specified for the following domain [*service* = “WSOL-EVERY”], then according to the rules for the WSOL constants, this domain is wrong. The value of domain attribute “*service*” of a service offering cannot be “WSOL-EVERY” since a service offering can only be specified for a particular Web Service. The WSOL parser detects and reports this error as shown in the following screen shot:

```

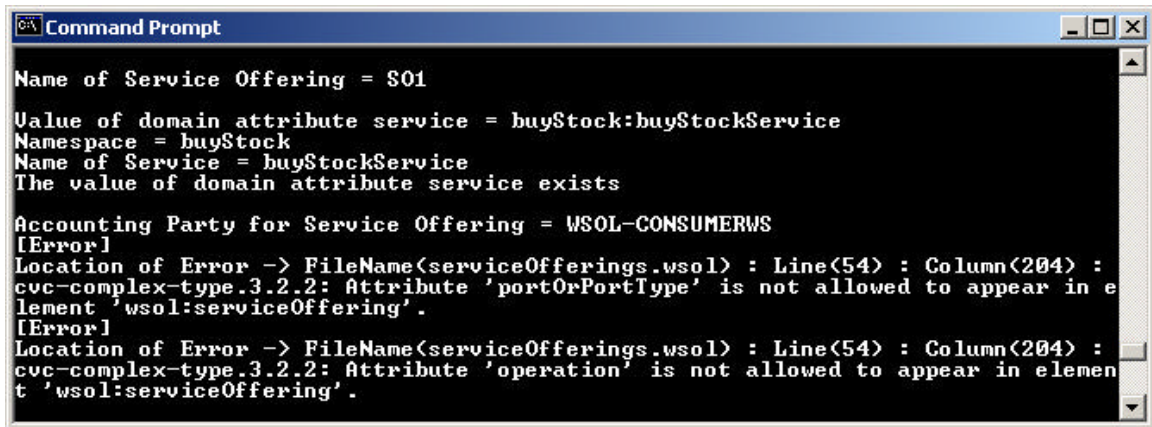
Command Prompt
Name of Service Offering = S01
Value of domain attribute service = WSOL-EVERY
[Error] serviceOfferings.wsol:59:58: The value of domain attribute service of Serv
ice Offering construct cannot be WSOL-EUERY in the value "WSOL-EUERY" of attribute
"service" on element "wsol:serviceOffering"
Accounting Party for Service Offering = WSOL-CONSUMERWS

```

**Figure 6.21 An error detected in a service offering**

3. The following error is also introduced in the service offering “SO1” mentioned earlier. If the service offering “SO1” is specified for the following domain [*service* = “buyStock:buyStockService”, *portOrPortType* = “buyStock:buyStockServicePort”, and *operation* = “buyStock:buySingleStockOperation”], then it is an error because the domain for

a service offering (in contrast to the other WSOL constructs) is determined only by the domain attribute “*service*”. The WSOL parser detects and reports this error as shown in the following screen shot:



```
Command Prompt
Name of Service Offering = S01
Value of domain attribute service = buyStock:buyStockService
Namespace = buyStock
Name of Service = buyStockService
The value of domain attribute service exists
Accounting Party for Service Offering = WSOL-CONSUMERWS
[Error]
Location of Error -> FileName(serviceOfferings.wsol) : Line(54) : Column(204) :
cvc-complex-type.3.2.2: Attribute 'portOrPortType' is not allowed to appear in element
'wsol:serviceOffering'.
[Error]
Location of Error -> FileName(serviceOfferings.wsol) : Line(54) : Column(204) :
cvc-complex-type.3.2.2: Attribute 'operation' is not allowed to appear in element
'wsol:serviceOffering'.
```

**Figure 6.22 An error detected in a service offering**

### 6.3 Summary

The ability of the WSOL parser to recognize WSOL statements and to detect and report syntax and semantic errors in them was demonstrated, using the buyStock Web Service example, in this chapter. Errors were introduced deliberately in the WSOL and the WSOL-related documents and it was observed that the WSOL parser could detect and report all of them. A few examples of the errors introduced, detected, and reported by the WSOL parser, along with their screen shots were also shown in this chapter.

## 7 Conclusions, Summary of Contributions and Future Work

---

### 7.1 Conclusions

Formal, explicit and precise specification of constraints, management statements, classes of service for Web Services and contracts between Web Services is essential for management of Web Services and Web Service compositions. WSOL (Web Service Offerings Language), a new XML-based language, fully compatible with WSDL, enables such specification.

When development of the WSOL started, there was no language for specification of constraints and classes of service for Web Services. Languages related to WSOL such as WSLA, WSML, DAML-S, etc. started being developed only in parallel to WSOL. These languages are still under development. From the comparison made between WSOL and the languages most relevant to WSOL (i.e., WSLA, WSML, and DAML-S), it was observed that none of the languages related to WSOL addressed all the issues addressed by WSOL and could replace WSOL.

WSOL was only a set of some concepts and requirements when I started working on my thesis. It did not have any grammar. Without a grammar, the WSOL language could not be used. WSOL documents conforming to the grammar could not be created. Hence, the WSOL language needed a grammar. In this thesis, the WSOL grammar was successfully developed using the XML Schema language as shown in Section 4.

The WSOL language also did not have any tool for validation of WSOL documents against the WSOL grammar and for detection and notification of eventual syntax and semantic errors in them. Therefore, the WSOL language needed a parser. The development of a parser for the WSOL language was also accomplished in this thesis as

shown in Section 5. The WSOL parser was developed by reusing the Apache Xerces2 Java Parser. The ability of the WSOL parser to validate WSOL and WSOL-related (i.e., WSDL and Ontology) documents and to detect and report syntax and semantic errors in them was also demonstrated in this thesis using an illustrative example in Section 6.

## 7.2 Summary of Contributions

The main contributions of this Master's thesis are:

- The XML grammar for the WSOL language has been developed using the XML Schema language, as shown in Section 4. Before this Master's thesis, WSOL was only a set of concepts and ideas without precise language grammar and without any example WSOL documents. The WSOL grammar can now be used to create WSOL documents conforming to the grammar. Development of the XML grammar for WSOL verifies that the WSOL language architecture and language constructs are feasible and implementable.
- A parser for the WSOL language, named "Premier" WSOL parser, has been developed as shown in Section 5. The WSOL parser implements WSOL concepts (Inheritance, Include mechanism, and Instantiate mechanism) and also detects and reports syntax and some semantic errors. The WSOL parser can now be used to validate WSOL documents and detect and report syntax and semantic errors in them. Development of the WSOL parser demonstrates that the developed WSOL grammar is well-formed.
- An illustrative WSOL example (the buyStock Web Service) has been developed as shown in Section 2.2.2. Using this example, the WSOL grammar and the WSOL

parser are verified. The ability of the WSOL parser to detect and report syntax and semantic errors in WSOL documents has also been demonstrated using this example, as shown in Section 6.

- The implementability of the WSOL language architecture and constructs has been verified. Based on the experiences with the WSOL grammar and the WSOL parser, frequent feedbacks have been provided about the implementability of WSOL language constructs and ideas have been generated to improve the WSOL language. In this manner, a very important contribution has been made to the development and improvement of the WSOL language.
- The initial results of the work on the WSOL language have been published in 3 refereed papers [40, 41, 50], as well as 7 non-refereed workshop position papers, research reports, and presentations [30, 42, 43, 44, 47, 48, 49]. The feedback received from these publications and presentations was very positive. This demonstrates that the research community considers the work on WSOL needed and of high quality. In addition to these past publications, we have recently submitted two full papers for publications at major international conferences [45, 46]. These papers are currently in the review process. We are currently working on a couple of submissions to international journals, as well as new conference papers (e.g., about the WSOL parser). In all these publications about WSOL, the work done in this Master's thesis has very important place.



### 7.3 Future Work

- a) Recently, WSOL was conceptually extended with several new concepts, such as periodic future-conditions and periodic QoS constraints. However, these recent improvements are not yet built into the XML grammar for WSOL and the WSOL parser. This has to be done as soon as possible.
- b) Currently, the WSOL language is compatible with the version 1.1 of the WSDL language. The version 1.2 of the WSDL language is being developed at present. When this development is completed, the WSOL language concepts, the WSOL grammar, and the WSOL parser will have to be updated accordingly.
- c) The WSOL Parser has to be improved in several ways:
  - The WSOL parser currently does not implement the checks for sub-domains. For every pair of an inner and an outer domain in a WSOL document, the WSOL parser needs to check whether the inner domain is a sub-domain of the outer domain.
  - The WSOL parser currently does not implement the checks for specialization of domains. For each inclusion or CGT instantiation statement specified in a WSOL document, the WSOL parser needs to check whether the new domain is the same as the old domain or a specialization of the old domain.
  - The WSOL language relies on external ontologies of QoS metrics, measurement units, and currency units. These ontologies, at present, consist of only a simple collection of names of QoS metrics, measurement units, and currency units, respectively. These ontological specifications need to be improved to include additional information. When these ontological specifications are improved, the

WSOL parser should also be improved to enable detection of more semantic errors related to QoS metrics, measurement units, and currency units.

- At present, the documents to be parsed have to be given to the WSOL parser in a specific order - the WSDL documents first, followed by the ontology documents, and finally the WSOL documents. In case of multiple WSDL or WSOL documents, the documents that are being imported in other documents have to be parsed first. All the documents to be parsed are currently arranged in an appropriate order manually. There is a need to enable that the WSOL parser can handle documents no matter in which order they are specified. One way to achieve this is to pre-process a list of input documents and automatically generate the appropriate parsing descriptor for the WSOL parser. A parsing descriptor would be a document that lists documents to be parsed in the correct order.
  - The WSOL parser currently does not validate specifications of dynamic relationships between service offerings. As discussed in Section 4.2, these specifications are described outside WSOL files in a special XML format. The WSOL parser has to be extended to be able to parse these documents.
- d) A full WSOL compiler has to be developed for the WSOL language. This compiler would use the already developed WSOL parser and would add automatic generation of constraint-checking code for run-time monitoring, metering, and evaluation of WSOL constraints.
- e) A Java API for easier generation of WSOL documents conforming to the WSOL grammar also has to be developed.

## 8 References

---

1. “Building bug-free O-O software: An introduction to Design by Contract™”, Article, Eiffel Software Inc., on-line at: <http://archive.eiffel.com/doc/manuals/technology/contract/>
2. “DTD Tutorial”, Free Web Building Tutorials, W3Schools.com, on-line at: <http://www.w3schools.com/dtd/default.asp>
3. “Namespaces in XML”, World Wide Web Consortium, January 1999, on-line at: <http://www.w3.org/TR/REC-xml-names/>
4. “UDDI Technical White Paper”, Publication, Ariba Inc., International Business Machines Corporation and Microsoft Corporation, September 6, 2000, on-line at: [http://www.uddi.com/pubs/Iru\\_UDDI\\_Technical\\_White\\_Paper.doc](http://www.uddi.com/pubs/Iru_UDDI_Technical_White_Paper.doc)
5. “XMethods”, WWW page, XMethod, Inc., 2002, on-line at: <http://www.xmethods.com/>
6. “XML Namespaces”, Free Web Building Tutorials, W3Schools.com, on-line at: [http://www.w3schools.com/xml/xml\\_namespaces.asp](http://www.w3schools.com/xml/xml_namespaces.asp)
7. “XML Parser for Java”, product information, alphaWorks emerging technologies, IBM, 2000, online at: <http://www.alphaworks.ibm.com/tech/xml4j>
8. Bedunah, John B. “XML: The Future Of The Web”, *ACM Crossroads*, Vol.6 Issue 2, Nov. 1999, pp.5-10.
9. Cerami, E. “Web Services Essentials”, Book, First Edition, O’Reilly & Associates, Inc., February 2002.
10. Christensen, E., Curbera, F., Meredith, G., Weerawarana S. “Web Services

- Description Language (WSDL) 1.1”, W3C Note, Ariba, International Business Machines Corporation and Microsoft, March 15, 2001, on-line at: <http://www.w3.org/TR/wsdl>
11. Costello, Roger L. “XML Schema Tutorial”, xFront.com (XML Technologies), Dec. 2002, on-line at: <http://www.xfront.com/>
  12. Curbera, F., Duftler, M., Khalaf, R., Mukhi, N., Nagy, W., Weerawarana, S. “Unraveling the Web Services Web - An Introduction to SOAP, WSDL, and UDDI”, *IEEE Internet Computing*, Vol.6 Issue 2, March-April 2002, pp.86-93.
  13. Curbera, F., Mukhi, N., Weerawarana, S. “On the Emergence of a Web Services Component Model”, In *Proc. of the WCOP’01 workshop at ECOOP’01*, Budapest, Hungary, June 2001, on-line at: <http://www.research.microsoft.com/users/cszypers/events/WCOP2001/>
  14. Dan, A., Franck, R., Keller, A., King, R., Ludwig, H. “Web Service Level Agreement (WSLA) Language Specification”, In *Documentation for Web Services Toolkit, version 3.2.1*, International Business Machines Corporation (IBM), August 2002.
  15. Ferguson, D. F. “Web Services Architecture: Direction and Position Paper”, In *Proc. of the W3C Workshop on Web Services – WSWS’01*, San Jose, USA, April 2001, W3C, on-line at: <http://www.w3c.org/2001/03/WSWS-popa/paper44>
  16. Florescu, D., Grunhagen, A., Kossmann, D. “XL: An XML Programming Language for Web Service Specification and Composition”, WWW page, 2002, on-line at: <http://www2002.org/CDROM/refereed/481/>
  17. Frolund, S., Koistinen, J. “QML: A language for Quality of Service

- Specification”, Technical Report, Software Technology Laboratory, Hewlett Packard, February 1998, on-line at: <http://www.hpl.hp.com/techreports/98/HPL-98-10.pdf>
18. Glass, G. “The Web services @evolution: Part 4, Web Services Description Language (WSDL)”, Web services articles, IBM developerWorks, February 2001, on-line at: <http://www-106.ibm.com/developerworks/webservices/library/ws-peer4/?dwzone=webservices>
19. Gottschalk, K., Graham, S., Kreger, H., Snell, J. “Introduction to Web services architecture”, *IBM Systems Journal*, Vol.41, No.2, 2002.
20. Irani, R. “An Introduction to ebXML”, *Web Services Architect Journal*, July 2001, on-line at: <http://www.webservicesarchitect.com/content/articles/irani02.asp>
21. Irani, R. “Part II - ebXML and Web Services”, *Web Services Architect Journal*, July 2001, on-line at: <http://www.webservicesarchitect.com/content/articles/irani03.asp>
22. Jacobsen, H. -A., Kramer, B. “Modeling Interface Definition Language Extensions”, In *Proc. Of the 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-37)*, Sydney, Australia, November 2000, on-line at: <http://www.fernuni-hagen.de/DVT/Publikationen/tools2000.pdf>
23. Jepsen, T. “SOAP Cleans up Interoperability Problems on the Web”, *IT Professional*, Vol.3 Issue 1, January-February 2001, pp.52-55.
24. Kamthan, P. “XML Namespaces: Universal Identification in XML Markup”,

Internet Related Technologies, November 1999, on-line at:  
<http://tech.irt.org/articles/js193/>

25. Keller, A., Ludwig, H. "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", IBM Research Report, International Business Machines Corporation (IBM), May 22, 2002.
26. Kulchenko, P., Snell, J., Tidwell, D. "Programming Web Services with SOAP", Book, First Edition, O'Reilly & Associates, Inc., January 2002.
27. Lear, A. C. "XML seen as integral to application integration", *IT Professional*, Vol.1 Issue 5, Sept-Oct. 1999, pp.12-16.
28. Marshall, I., Mckee, P. "Behavioural Specification Using XML", In *Proc. Of the 7<sup>th</sup> IEEE Workshop on Future Trends of Distributed Computing Systems – FTDCS'99*, Cape Town, South Africa, December 1999, IEEE Computer Society Press, pp. 53-59.
29. Ogbuchi, Uche. "Using WSDL in SOAP applications, An introduction to WSDL for SOAP programmers", Web services articles, IBM developerWorks, November 2000, on-line at: <http://www-106.ibm.com/developerworks/webservices/library/ws-soap/index.html?dwzone=ws>
30. Patel, K., Tasic, V., Pagurek, B. "XML Grammar and Parser for WSOL", In *Proc. of the CITO Knowledge Network Conference 2002*, Ottawa, Canada, October 2002.
31. Ramanujan, A., Roy, J. "XML: Data's Universal Language", *IT Professional*, Vol.2 Issue 3, May-June 2000, pp.32-36.

32. Sahai, A. et al. "Automated SLA Monitoring for Web Services", IEEE/IFIP DSOM 2002, Montreal, Canada, Oct. 2002.
33. Sahai, A., Durante, A., Machiraju, V. "Towards Automated SLA Management for Web Services", Research Report HPL-2001-310 (R.1), Hewlett-Packard Laboratories Palo Alto, July 2002, on-line at: <http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf>
34. Shohoud, Y. "Introduction to WSDL", devxpert.com, on-line at: <http://www.devxpert.com/tutors/wsdl/wsdl.asp>
35. Skonnard, A. "Understanding XML Namespaces", *MSDN Magazine*, July 2001, on-line at: <http://msdn.microsoft.com/msdnmag/issues/01/07/xml/xml0107.asp>
36. The Apache Software Foundation. "Xerces Java Parser", product information, The Apache Software Foundation, 2000, online at: <http://xml.apache.org/xerces-j/index.html>
37. The Apache Software Foundation. "Xerces2 Java Parser", product information, The Apache Software Foundation, 2002, online at: <http://xml.apache.org/xerces2-j/index.html>
38. The DAML Services Coalition. "DAML-S: Semantic Markup for Web Services", WWW page, June 2001, on-line at: <http://www.daml.org/services/damls/2001/05/daml-s.html>
39. The DAML-S Coalition. "DAML-S: Web Service Description for the Semantic Web", In *Proc. of the International Semantic Web Conference*, Sardinia, Italia, June 2002, on-line at: <http://www.icsi.berkeley.edu/~snarayan/ISWC2002.pdf>

40. Tasic, V., Esfandiari, B., Pagurek, B., Patel, K. “On Requirements for Ontologies in Management of Web Services”, In *Proc. of the Workshop on Web Services, e-Business, and the Semantic Web - WES* (at CaiSE'02), Toronto, Canada, May 2002.
41. Tasic, V., Pagurek, B., Esfandiari, B., Patel, K. “Management of Compositions of E- and M-Business Web Services with Multiple Classes of Service”, In *Proc. of NOMS (Network Operations and Management Symposium) 2002*, Florence, Italy, April 15-19, 2002.
42. Tasic, V., Pagurek, B., Esfandiari, B., Patel, K. “On the Management of Compositions of Web Services”, Position paper at the *Workshop on Object-Oriented Web Services - OOWS* (at OOPSLA 2001), Tampa, USA, October 2001.
43. Tasic, V., Pagurek, B., Esfandiari, B., Patel, K. “On Various Approaches to Dynamic Adaptation of Distributed Component Compositions”, *Technical Report OCIECE-02-02*, Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE), June 2002.
44. Tasic, V., Pagurek, B., Esfandiari, B., Patel, K., Ma, W. “Web Service Offerings language (WSOL) and Web Service Composition Management (WSCM)”, In *Proc. of the Object-Oriented Web Services Workshop at OOPSLA 2002*, Seattle, USA.
45. Tasic, V., Pagurek, B., Patel, K. “WSOL – A Language for the Formal Specification of Various Constraints and Classes of Service for Web Services”, *Technical Report OCIECE-02-06*, Ottawa-Carleton Institute for Electrical and Computer Engineering - OCIECE, Ottawa, Canada, November 2002.



46. Tasic, V., Pagurek, B., Patel, K., Esfandiari, B., Ma, W. "Management Applications of the Web Service Offerings Language (WSOL)", submitted for conference publication on November 30, 2002.
47. Tasic, V., Patel, B., Pagurek, B. "Characteristics of the Web Service Offerings Language (WSOL)", presented at the *Network Management and Artificial Intelligence Lab Seminar*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, November 2002.
48. Tasic, V., Patel, B., Pagurek, B., Esfandiari, B. "Dynamic Adaptation of Service Compositions Using Service Offerings", presented at the *Network Management and Artificial Intelligence Lab Seminar*, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, February 2002.
49. Tasic, V., Patel, K., Pagurek, B. "E-Business Service Components with Multiple Classes of Service and Dynamic Adaptability Mechanisms", In *Proc. of the CITO Knowledge Network Conference 2001*, Ottawa, Canada, October 2001.
50. Tasic, V., Patel, K., Pagurek, B. "Web Service Offerings Language", In *Proc. of the Workshop on Web Services, e-Business, and the Semantic Web at CaiSE'02*, Toronto, Canada, May 2002.
51. Weerawarna, S. "Web Services and Software Engineering: Challenges and Opportunities", presented at the International Conference on Software Engineering (ICSE) 2001, Toronto, Canada, May 2001, on-line at: <http://www.csr.uvic.ca/icse2001/schedule.html#s15>
52. World Wide Web Consortium (W3C). "Mathematical Markup Language (MathML) 1.01 Specification", W3C Recommendation, July 7, 1999, on-line at:

<http://www.w3.org/TR/REC-MathML/>

53. World Wide Web Consortium (W3C). “Web Services Architecture”, W3C Working Draft, Nov. 14, 2002, on-line at: <http://www.w3.org/TR/2002/WD-ws-arch-20021114/>

54. World Wide Web Consortium (W3C). “XML Path Language (XPath) Version 1.0”, W3C Recommendation, Nov. 16, 1999, on-line at: <http://www.w3.org/TR/xpath#section-Introduction>

## Appendix A: WSOL Schema

---

```
<?xml version = "1.0" encoding = "UTF-8"?>

<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://www.sce.carleton.ca/~kpatel/WSOL/WSOLSchema"
  xmlns:wsol = "http://www.sce.carleton.ca/~kpatel/WSOL/WSOLSchema"
  xmlns:expressionSchema =
"http://www.sce.carleton.ca/~kpatel/WSOL/ExpressionSchema"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
  elementFormDefault = "qualified"
  attributeFormDefault = "unqualified">

  <import namespace = http://www.sce.carleton.ca/~kpatel/WSOL/ExpressionSchema
    schemaLocation = "ExpressionSchema.xsd"/>

  <element name = "WSOLdefinitions" type = "wsol:WSOLdefinitionsType"/>
  <complexType name = "WSOLdefinitionsType">
    <sequence>
      <element ref = "wsol:import" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:externalOperationCall" minOccurs = "0" maxOccurs
        = "unbounded"/>
      <element ref = "wsol:constraint" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:subscription" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:price" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:priceDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:penalty" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:penaltyDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:managementResponsibility" minOccurs = "0"
        maxOccurs = "unbounded"/>
      <element ref = "wsol:statement" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:include" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:CG" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:CGT" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:instantiate" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:serviceOffering" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <any namespace = "##other" processContents = "strict" minOccurs = "0"
        maxOccurs = "unbounded"/>
    </sequence>
    <attribute name = "targetNamespace" type = "anyURI"/>
  </complexType>
```

```

<element name = "import" type = "wsol:importType"/>
<complexType name = "importType">
  <attribute name = "namespace" use = "required" type = "anyURI"/>
  <attribute name = "location" use = "required" type = "anyURI"/>
</complexType>

<element name = "serviceOffering">
  <complexType>
    <sequence>
      <element ref = "wsol:externalOperationCall" minOccurs = "0"
        maxOccurs = "unbounded"/>
      <element ref = "wsol:constraint" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:subscription" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:price" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:priceDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:penalty" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:penaltyDefault" minOccurs = "0"
        maxOccurs = "unbounded"/>
      <element ref = "wsol:managementResponsibility" minOccurs =
        "0" maxOccurs = "unbounded"/>
      <element ref = "wsol:statement" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:include" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:CG" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:instantiate" minOccurs = "0" maxOccurs =
        "unbounded"/>
    </sequence>
    <attribute name = "name" use = "required" type = "NCName"/>
    <attribute name = "service" use = "required" type = "QName"/>
    <attribute name = "extends" default = "WSOL-NONE" type = "QName"/>
    <attribute name = "accountingParty" default = "WSOL-SUPPLIERWS"
      type = "anyURI"/>
  </complexType>
</element>

<element name = "externalOperationCall">
  <complexType>
    <sequence>
      <element ref = "wsol:callList" minOccurs = "0" maxOccurs =
        "1"/>
    </sequence>
    <attribute name = "callID" use = "required" type = "NCName"/>
    <attribute name = "calledOperDescription" use = "required" type =
      "QName"/>
    <attribute name = "calledOperImplementation" use = "required" type =
      "QName"/>
    <attribute name = "service" use = "required" type = "QName"/>
    <attribute name = "portOrPortType" use = "required" type = "QName"/>
    <attribute name = "operation" use = "required" type = "QName"/>
  </complexType>
</element>

```

```

        </complexType>
    </element>

    <element name = "callList">
        <complexType>
            <sequence>
                <element ref = "wsol:parameterValue" minOccurs = "0"
                    maxOccurs = "unbounded"/>
            </sequence>
        </complexType>
    </element>

    <element name = "parameterValue">
        <complexType>
            <sequence>
                <element ref = "wsol:partName"/>
                <element ref = "wsol:paramValue"/>
            </sequence>
        </complexType>
    </element>

    <element name = "partName">
        <complexType>
            <attribute name = "name" use = "required" type = "QName"/>
        </complexType>
    </element>

    <element name = "paramValue">
        <complexType>
            <choice>
                <element ref = "expressionSchema:arithmeticExpression"/>
                <element ref = "expressionSchema:stringExpression"/>
            </choice>
        </complexType>
    </element>

    <element name = "constraint" type = "wsol:constraintType"/>
    <complexType name = "constraintType">
        <attribute name = "name" use = "required" type = "NCName"/>
        <attribute name = "service" use = "required" type = "QName"/>
        <attribute name = "portOrPortType" use = "required" type = "QName"/>
        <attribute name = "operation" use = "required" type = "QName"/>
    </complexType>

    <element name = "statement">
        <complexType>
            <attribute name = "name" use = "required" type = "NCName"/>
        </complexType>
    </element>

    <element name = "CG">
        <complexType>
            <sequence>
                <element ref = "wsol:externalOperationCall" minOccurs = "0"
                    maxOccurs = "unbounded"/>
            </sequence>
        </complexType>
    </element>

```

```

        <element ref = "wsol:constraint" minOccurs = "0" maxOccurs =
        "unbounded"/>
        <element ref = "wsol:subscription" minOccurs = "0" maxOccurs =
        "unbounded"/>
        <element ref = "wsol:price" minOccurs = "0" maxOccurs =
        "unbounded"/>
        <element ref = "wsol:priceDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
        <element ref = "wsol:penalty" minOccurs = "0" maxOccurs =
        "unbounded"/>
        <element ref = "wsol:penaltyDefault" minOccurs = "0"
        maxOccurs = "unbounded"/>
        <element ref = "wsol:managementResponsibility" minOccurs =
        "0"/>
        <element ref = "wsol:statement" minOccurs = "0" maxOccurs =
        "unbounded"/>
        <element ref = "wsol:include" minOccurs = "0" maxOccurs =
        "unbounded"/>
        <element ref = "wsol:CG" minOccurs = "0" maxOccurs =
        "unbounded"/>
        <element ref = "wsol:instantiate" minOccurs = "0" maxOccurs =
        "unbounded"/>
    </sequence>
    <attribute name = "name" use = "required" type = "NCName"/>
    <attribute name = "extends" use = "optional" default = "WSOL-NONE"
    type = "QName"/>
    <attribute name = "service" use = "required" type = "QName"/>
    <attribute name = "portOrPortType" use = "required" type = "QName"/>
    <attribute name = "operation" use = "required" type = "QName"/>
</complexType>
</element>

<element name = "instantiate">
    <complexType>
        <sequence>
            <element ref = "wsol:parmValue" maxOccurs = "unbounded"/>
        </sequence>
        <attribute name = "CGTName" use = "required" type = "QName"/>
        <attribute name = "resService" use = "required" type = "QName"/>
        <attribute name = "resPortOrPortType" use = "required" type =
        "QName"/>
        <attribute name = "resOperation" use = "required" type = "QName"/>
        <attribute name = "resCGName" use = "required" type = "NCName"/>
    </complexType>
</element>

<element name = "parmValue">
    <complexType>
        <sequence>
            <element ref = "wsol:numberWithUnitConstant" minOccurs =
            "0"/>
        </sequence>
        <attribute name = "name" use = "required" type = "QName"/>
        <attribute name = "value" type = "QName"/>
    </complexType>
</element>

```

```

<element name = "include">
  <complexType>
    <attribute name = "constructName" use = "required" type = "QName"/>
    <attribute name = "resService" use = "required" type = "QName"/>
    <attribute name = "resPortOrPortType" use = "required" type =
      "QName"/>
    <attribute name = "resOperation" use = "required" type = "QName"/>
    <attribute name = "resName" use = "required" type = "NCName"/>
  </complexType>
</element>

<element name = "CGT">
  <complexType>
    <sequence>
      <element ref = "wsol:parameter" minOccurs = "1" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:externalOperationCall" minOccurs = "0"
        maxOccurs = "unbounded"/>
      <element ref = "wsol:constraint" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:subscription" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:price" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:priceDefault" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:penalty" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:penaltyDefault" minOccurs = "0"
        maxOccurs = "unbounded"/>
      <element ref = "wsol:managementResponsibility" minOccurs =
        "0"/>
      <element ref = "wsol:statement" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:include" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:CG" minOccurs = "0" maxOccurs =
        "unbounded"/>
      <element ref = "wsol:instantiate" minOccurs = "0" maxOccurs =
        "unbounded"/>
    </sequence>
    <attribute name = "name" use = "required" type = "NCName"/>
    <attribute name = "extends" default = "WSOL-NONE" type = "QName"/>
    <attribute name = "service" use = "required" type = "QName"/>
    <attribute name = "portOrPortType" use = "required" type = "QName"/>
    <attribute name = "operation" use = "required" type = "QName"/>
  </complexType>
</element>

<element name = "parameter">
  <complexType>

    <attribute name = "name" use = "required" type = "NCName"/>
    <attribute name = "dataType" use = "required" type = "QName"/>
    <attribute name = "unit" default = "WSOL-NOUNIT" type = "QName"/>

```

```

        </complexType>
    </element>

    <element name = "subscription">
        <complexType>
            <sequence>
                <element ref = "wsol:numberWithUnitConstant"/>
                <element ref = "wsol:subscriptionDuration"/>
            </sequence>
        </complexType>
    </element>

    <element name = "priceDefault">
        <complexType>
            <sequence>
                <element ref = "wsol:numberWithUnitConstant"/>
            </sequence>
        </complexType>
    </element>

    <element name = "penaltyDefault">
        <complexType>
            <sequence>
                <element ref = "wsol:numberWithUnitConstant"/>
            </sequence>
        </complexType>
    </element>

    <element name = "price">
        <complexType>
            <sequence>
                <element ref = "wsol:numberWithUnitConstant"/>
            </sequence>
            <attribute name = "name" use = "required" type = "NCName"/>
            <attribute name = "service" use = "required" type = "QName"/>
            <attribute name = "portOrPortType" use = "required" type = "QName"/>
            <attribute name = "operation" use = "required" type = "QName"/>
        </complexType>
    </element>

    <element name = "penalty">
        <complexType>
            <sequence>
                <element ref = "wsol:numberWithUnitConstant"/>
            </sequence>
            <attribute name = "name" use = "required" type = "NCName"/>
            <attribute name = "service" use = "required" type = "QName"/>
            <attribute name = "portOrPortType" use = "required" type = "QName"/>
            <attribute name = "operation" use = "required" type = "QName"/>
        </complexType>
    </element>

    <element name = "subscriptionDuration">
        <complexType>
            <attribute name = "duration" use = "required" type = "duration"/>

```



```

        </complexType>
    </element>

    <complexType name = "numberWithUnit">
        <sequence>
            <element ref = "wsol:number"/>
            <element ref = "wsol:unit"/>
        </sequence>
    </complexType>

    <element name = "unit">
        <complexType>
            <attribute name = "type" use = "required" type = "QName"/>
        </complexType>
    </element>

    <element name = "number">
        <complexType>
            <attribute name = "value" use = "required" type = "double"/>
        </complexType>
    </element>

    <element name = "numberWithUnitConstant" type = "wsol:numberWithUnit"/>
    <element name = "managementResponsibility">
        <complexType>
            <sequence>
                <element ref = "wsol:supplierResponsibility" minOccurs = "0"
                    maxOccurs = "unbounded"/>
                <element ref = "wsol:consumerResponsibility" minOccurs = "0"
                    maxOccurs = "unbounded"/>
                <element ref = "wsol:independentResponsibility" minOccurs =
                    "0" maxOccurs = "unbounded"/>
            </sequence>
            <attribute name = "name" use = "required" type = "NCName"/>
        </complexType>
    </element>

    <element name = "supplierResponsibility">
        <complexType>
            <attribute name = "scope" use = "required" type = "QName"/>
        </complexType>
    </element>

    <element name = "consumerResponsibility">
        <complexType>
            <attribute name = "scope" use = "required" type = "QName"/>
        </complexType>
    </element>

    <element name = "independentResponsibility">
        <complexType>
            <attribute name = "scope" use = "required" type = "QName"/>
            <attribute name = "entity" use = "required" type = "anyURI"/>
        </complexType>
    </element>
</schema>

```

## Appendix B: Expression Schema

---

```
<?xml version = "1.0" encoding = "UTF-8"?>

<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://www.sce.carleton.ca/~kpatel/W SOL/ExpressionSchema"
  xmlns:expressionSchema =
"http://www.sce.carleton.ca/~kpatel/W SOL/ExpressionSchema"
  xmlns:wsol = "http://www.sce.carleton.ca/~kpatel/W SOL/W SOLSchema"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
  elementFormDefault = "qualified"
  attributeFormDefault = "unqualified">

  <import namespace = "http://www.sce.carleton.ca/~kpatel/W SOL/W SOLSchema"
  schemaLocation = "WSOLSchema.xsd"/>

  <element name = "booleanExpression" type =
"expressionSchema:booleanExpressionType"/>
  <complexType name = "booleanExpressionType">
    <choice>
      <element ref = "expressionSchema:booleanConstant"/>
      <element ref = "expressionSchema:booleanVariable"/>
      <element ref = "expressionSchema:externalOperationResult"/>
      <element ref = "expressionSchema:booleanExpression"/>
      <element ref = "expressionSchema:quantifiedExpression"/>
      <sequence>
        <element ref = "expressionSchema:unaryBooleanOperator"/>
        <element ref = "expressionSchema:booleanExpression"/>
      </sequence>
      <sequence>
        <element ref = "expressionSchema:booleanExpression"/>
        <element ref = "expressionSchema:binaryBooleanOperator"/>
        <element ref = "expressionSchema:booleanExpression"/>
      </sequence>
      <sequence>
        <element ref = "expressionSchema:booleanExpression"/>
        <element ref = "expressionSchema:booleanComparator"/>
        <element ref = "expressionSchema:booleanExpression"/>
      </sequence>
      <sequence>
        <element ref = "expressionSchema:arithmeticExpression"/>
        <element ref = "expressionSchema:arithmeticComparator"/>
        <element ref = "expressionSchema:arithmeticExpression"/>
      </sequence>
      <sequence>
        <element ref =
"expressionSchema:arithmeticWithUnitExpression"/>
        <element ref =
"expressionSchema:arithmeticWithUnitComparator"/>
        <element ref =
"expressionSchema:arithmeticWithUnitExpression"/>
      </sequence>
    </choice>
  </complexType>

```

```

        </sequence>
    <sequence>
        <element ref = "expressionSchema:stringExpression"/>
        <element ref = "expressionSchema:stringComparator"/>
        <element ref = "expressionSchema:stringExpression"/>
    </sequence>
</sequence>
</choice>
</complexType>

<element name = "booleanConstant" type =
"expressionSchema:booleanConstantType"/>
<complexType name = "booleanConstantType">
    <attribute name = "type" use = "required" type =
        "expressionSchema:booleanConstantTypeChoice"/>
</complexType>

<simpleType name = "booleanConstantTypeChoice">
    <restriction base = "string">
        <enumeration value = "true"/>
        <enumeration value = "false"/>
    </restriction>
</simpleType>

<simpleType name = "unaryBooleanOperatorTypeChoice">
    <restriction base = "string">
        <enumeration value = "NOT"/>
    </restriction>
</simpleType>

<simpleType name = "binaryBooleanOperatorTypeChoice">
    <restriction base = "string">
        <enumeration value = "AND"/>
        <enumeration value = "OR"/>
        <enumeration value = "XOR"/>
        <enumeration value = "EQUIVALENT"/>
        <enumeration value = "IMPLIES"/>
    </restriction>
</simpleType>

<simpleType name = "booleanComparatorTypeChoice">
    <restriction base = "string">
        <enumeration value = "==" />
        <enumeration value = "!=" />
    </restriction>
</simpleType>

<simpleType name = "quantifiedExpressionTypeChoice">
    <restriction base = "string">
        <enumeration value = "FOR_ALL_IN"/>
        <enumeration value = "EXISTS_IN"/>
    </restriction>

```

```

</simpleType>

<simpleType name = "unaryArithmeticOperatorTypeChoice">
  <restriction base = "string">
    <enumeration value = "-"/>
  </restriction>
</simpleType>

<simpleType name = "binaryArithmeticOperatorTypeChoice">
  <restriction base = "string">
    <enumeration value = "+"/>
    <enumeration value = "-"/>
    <enumeration value = "*"/>
    <enumeration value = "/">
    <enumeration value = "**"/>
  </restriction>
</simpleType>

<simpleType name = "arithmeticComparatorTypeChoice">
  <restriction base = "string">
    <enumeration value = "<"/>
    <enumeration value = ">"/>
    <enumeration value = "="/>
    <enumeration value = "<=""/>
    <enumeration value = ">=""/>
    <enumeration value = "!=""/>
  </restriction>
</simpleType>

<simpleType name = "arithmeticWithUnitComparatorTypeChoice">
  <restriction base = "string">
    <enumeration value = "<"/>
    <enumeration value = ">"/>
    <enumeration value = "="/>
    <enumeration value = "<=""/>
    <enumeration value = ">=""/>
    <enumeration value = "!=""/>
  </restriction>
</simpleType>

<simpleType name = "stringComparatorTypeChoice">
  <restriction base = "string">
    <enumeration value = "<"/>
    <enumeration value = ">"/>
    <enumeration value = "="/>
    <enumeration value = "<=""/>
    <enumeration value = ">=""/>
    <enumeration value = "!=""/>
  </restriction>
</simpleType>

<simpleType name = "timeOperatorTypeChoice">
  <restriction base = "string">
    <enumeration value = "BEFORE"/>
    <enumeration value = "AFTER"/>
    <enumeration value = "DURING"/>
  </restriction>
</simpleType>

```

```

        <enumeration value = "BETWEEN"/>
    </restriction>
</simpleType>

<simpleType name = "timeComparatorTypeChoice">
    <restriction base = "string">
        <enumeration value = "&lt;"/>
        <enumeration value = ">"/>
        <enumeration value = "==">
        <enumeration value = "&lt;=">
        <enumeration value = ">=">
        <enumeration value = "!=">
    </restriction>
</simpleType>

<simpleType name = "binaryArithmeticWithUnitOperator1TypeChoice">
    <restriction base = "string">
        <enumeration value = "+">
        <enumeration value = "-">
    </restriction>
</simpleType>

<simpleType name = "binaryArithmeticWithUnitOperator2TypeChoice">
    <restriction base = "string">
        <enumeration value = "*">
        <enumeration value = "/">
    </restriction>
</simpleType>

<element name = "externalOperationResult" type =
"expressionSchema:externalOperationResultType"/>
<complexType name = "externalOperationResultType">
    <attribute name = "callID" use = "required" type = "NCName"/>
    <attribute name = "resultPartName" use = "required" type = "QName"/>
</complexType>

<element name = "unaryBooleanOperator" type =
"expressionSchema:unaryBooleanOperatorType"/>
<complexType name = "unaryBooleanOperatorType">
    <attribute name = "type" use = "required" type =
"expressionSchema:unaryBooleanOperatorTypeChoice"/>
</complexType>

<element name = "binaryBooleanOperator" type =
"expressionSchema:binaryBooleanOperatorType"/>
<complexType name = "binaryBooleanOperatorType">
    <attribute name = "type" use = "required" type =
"expressionSchema:binaryBooleanOperatorTypeChoice"/>
</complexType>

<element name = "booleanComparator" type =
"expressionSchema:booleanComparatorType"/>
<complexType name = "booleanComparatorType">
    <attribute name = "type" use = "required" type =
"expressionSchema:booleanComparatorTypeChoice"/>
</complexType>

```

```

<element name = "quantifiedExpression" type =
"expressionSchema:quantifiedExpressionType"/>
<complexType name = "quantifiedExpressionType">
  <sequence>
    <element ref = "expressionSchema:booleanExpression"/>
  </sequence>
  <attribute name = "type" use = "required" type =
"expressionSchema:quantifiedExpressionTypeChoice"/>
  <attribute name = "arrayVariableName" use = "required" type = "QName"/>
</complexType>

<element name = "arithmeticExpression" type =
"expressionSchema:arithmeticExpressionType"/>
<complexType name = "arithmeticExpressionType">
  <choice>
    <element ref = "expressionSchema:arithmeticConstant"/>
    <element ref = "expressionSchema:arithmeticVariable"/>
    <element ref = "expressionSchema:externalOperationResult"/>
    <element ref = "expressionSchema:arithmeticExpression"/>
    <sequence>
      <element ref = "expressionSchema:unaryArithmeticOperator"/>
      <element ref = "expressionSchema:arithmeticExpression"/>
    </sequence>
    <sequence>
      <element ref = "expressionSchema:arithmeticExpression"/>
      <element ref = "expressionSchema:binaryArithmeticOperator"/>
      <element ref = "expressionSchema:arithmeticExpression"/>
    </sequence>
  </choice>
</complexType>

<element name = "arithmeticConstant" type =
"expressionSchema:arithmeticConstantType"/>
<complexType name = "arithmeticConstantType">
  <choice>
    <element ref = "expressionSchema:integerConstant"/>
    <element ref = "expressionSchema:floatConstant"/>
    <element ref = "expressionSchema:doubleConstant"/>
    <element ref = "expressionSchema:longConstant"/>
  </choice>
</complexType>

<element name = "QoSmetric" type = "expressionSchema:QoSmetricType"/>
<complexType name = "QoSmetricType">
  <attribute name = "metricType" use = "required" type = "QName"/>
  <attribute name = "metricUnit" use = "required" type = "QName"/>
  <attribute name = "service" use = "required" type = "QName"/>
  <attribute name = "portOrPortType" use = "required" type = "QName"/>
  <attribute name = "operation" use = "required" type = "QName"/>
  <attribute name = "measuredBy" use = "required" type = "QName"/>
</complexType>

<element name = "arithmeticWithUnitExpression" type =
"expressionSchema:arithmeticWithUnitExpressionType"/>
<complexType name = "arithmeticWithUnitExpressionType">

```

```

    <choice>
      <element ref = "wsol:numberWithUnitConstant"/>
      <element ref = "expressionSchema:arithmeticWithUnitVariable"/>
      <element ref = "expressionSchema:QoSmetric"/>
      <sequence>
        <element ref =
          "expressionSchema:arithmeticWithUnitExpression"/>
        <element ref =
          "expressionSchema:binaryArithmeticWithUnitOperator1"/>
        <element ref =
          "expressionSchema:arithmeticWithUnitExpression"/>
      </sequence>
      <sequence>
        <element ref =
          "expressionSchema:arithmeticWithUnitExpression"/>
        <element ref =
          "expressionSchema:binaryArithmeticWithUnitOperator2"/>
        <element ref = "expressionSchema:arithmeticConstant"/>
      </sequence>
    </choice>
  </complexType>

  <element name = "unaryArithmeticOperator" type =
    "expressionSchema:unaryArithmeticOperatorType"/>
  <complexType name = "unaryArithmeticOperatorType">
    <attribute name = "type" use = "required" type =
      "expressionSchema:unaryArithmeticOperatorTypeChoice"/>
  </complexType>

  <element name = "binaryArithmeticOperator" type =
    "expressionSchema:binaryArithmeticOperatorType"/>
  <complexType name = "binaryArithmeticOperatorType">
    <attribute name = "type" use = "required" type =
      "expressionSchema:binaryArithmeticOperatorTypeChoice"/>
  </complexType>

  <element name = "binaryArithmeticWithUnitOperator1" type =
    "expressionSchema:binaryArithmeticWithUnitOperator1Type"/>
  <complexType name = "binaryArithmeticWithUnitOperator1Type">
    <attribute name = "type" use = "required" type =
      "expressionSchema:binaryArithmeticWithUnitOperator1TypeChoice"/>
  </complexType>

  <element name = "binaryArithmeticWithUnitOperator2" type =
    "expressionSchema:binaryArithmeticWithUnitOperator2Type"/>
  <complexType name = "binaryArithmeticWithUnitOperator2Type">
    <attribute name = "type" use = "required" type =
      "expressionSchema:binaryArithmeticWithUnitOperator2TypeChoice"/>
  </complexType>

  <element name = "arithmeticComparator" type =
    "expressionSchema:arithmeticComparatorType"/>
  <complexType name = "arithmeticComparatorType">
    <attribute name = "type" use = "required" type =
      "expressionSchema:arithmeticComparatorTypeChoice"/>
  </complexType>

```

```

<element name = "arithmeticWithUnitComparator" type =
"expressionSchema:arithmeticWithUnitComparatorType"/>
<complexType name = "arithmeticWithUnitComparatorType">
  <attribute name = "type" use = "required" type =
"expressionSchema:arithmeticWithUnitComparatorTypeChoice"/>
</complexType>

<element name = "stringExpression" type = "expressionSchema:stringExpressionType"/>
<complexType name = "stringExpressionType">
  <choice>
    <element ref = "expressionSchema:stringConstant"/>
    <element ref = "expressionSchema:stringVariable"/>
    <element ref = "expressionSchema:externalOperationResult"/>
    <element ref = "expressionSchema:stringExpression"/>
  </choice>
</complexType>

<element name = "stringComparator" type =
"expressionSchema:stringComparatorType"/>
<complexType name = "stringComparatorType">
  <attribute name = "type" use = "required" type =
"expressionSchema:stringComparatorTypeChoice"/>
</complexType>

<element name = "timeExpression" type = "expressionSchema:timeExpressionType"/>
<complexType name = "timeExpressionType">
  <choice>
    <element ref = "expressionSchema:timeConstant"/>
    <element ref = "expressionSchema:timeVariable"/>
    <element ref = "expressionSchema:externalOperationResult"/>
    <element ref = "expressionSchema:timeExpression"/>
    <sequence>
      <element ref = "expressionSchema:timeOperator"/>
      <element ref = "expressionSchema:timeExpression"/>
    </sequence>
  </choice>
</complexType>

<element name = "timeConstant" type = "expressionSchema:timeConstantType"/>
<complexType name = "timeConstantType">
  <choice>
    <element ref = "expressionSchema:time"/>
    <element ref = "expressionSchema:date"/>
    <element ref = "expressionSchema:date_and_time"/>
    <element ref = "expressionSchema:time_duration"/>
    <element ref = "expressionSchema:gregorianYear"/>
    <element ref = "expressionSchema:gregorianYearMonth"/>
    <element ref = "expressionSchema:gregorianMonthDay"/>
    <element ref = "expressionSchema:gregorianMonth"/>
    <element ref = "expressionSchema:gregorianDay"/>
  </choice>
</complexType>

<element name = "timeOperator" type = "expressionSchema:timeOperatorType"/>
<complexType name = "timeOperatorType">

```



```

        <attribute name = "type" use = "required" type =
        "expressionSchema:timeOperatorTypeChoice"/>
    </complexType>

    <element name = "timeComparator" type = "expressionSchema:timeComparatorType"/>
    <complexType name = "timeComparatorType">
        <attribute name = "type" use = "required" type =
        "expressionSchema:timeComparatorTypeChoice"/>
    </complexType>

    <element name = "integerConstant" type = "expressionSchema:integerConstantType"/>
    <complexType name = "integerConstantType">
        <attribute name = "value" use = "required" type = "int"/>
    </complexType>

    <element name = "floatConstant" type = "expressionSchema:floatConstantType"/>
    <complexType name = "floatConstantType">
        <attribute name = "value" use = "required" type = "float"/>
    </complexType>

    <element name = "doubleConstant" type = "expressionSchema:doubleConstantType"/>
    <complexType name = "doubleConstantType">
        <attribute name = "value" use = "required" type = "double"/>
    </complexType>

    <element name = "longConstant" type = "expressionSchema:longConstantType"/>
    <complexType name = "longConstantType">
        <attribute name = "value" use = "required" type = "long"/>
    </complexType>

    <element name = "time" type = "expressionSchema:timeType"/>
    <complexType name = "timeType">
        <attribute name = "value" use = "required" type = "time"/>
    </complexType>

    <element name = "date" type = "expressionSchema:dateType"/>
    <complexType name = "dateType">
        <attribute name = "value" use = "required" type = "date"/>
    </complexType>

    <element name = "date_and_time" type = "expressionSchema:date_and_timeType"/>
    <complexType name = "date_and_timeType">
        <attribute name = "value" use = "required" type = "dateTime"/>
    </complexType>

    <element name = "time_duration" type = "expressionSchema:time_durationType"/>
    <complexType name = "time_durationType">
        <attribute name = "value" use = "required" type = "duration"/>
    </complexType>

    <element name = "gregorianYear" type = "expressionSchema:gregorianYearType"/>
    <complexType name = "gregorianYearType">
        <attribute name = "value" use = "required" type = "gYear"/>
    </complexType>

```

```

<element name = "gregorianYearMonth" type =
"expressionSchema:gregorianYearMonthType"/>
<complexType name = "gregorianYearMonthType">
  <attribute name = "value" use = "required" type = "gYearMonth"/>
</complexType>

<element name = "gregorianMonthDay" type =
"expressionSchema:gregorianMonthDayType"/>
<complexType name = "gregorianMonthDayType">
  <attribute name = "value" use = "required" type = "gMonthDay"/>
</complexType>

<element name = "gregorianMonth" type = "expressionSchema:gregorianMonthType"/>
<complexType name = "gregorianMonthType">
  <attribute name = "value" use = "required" type = "gMonth"/>
</complexType>

<element name = "gregorianDay" type = "expressionSchema:gregorianDayType"/>
<complexType name = "gregorianDayType">
  <attribute name = "value" use = "required" type = "gDay"/>
</complexType>

<element name = "booleanVariable" type = "expressionSchema:booleanVariableType"/>
<complexType name = "booleanVariableType">
  <attribute name = "bvName" use = "required" type = "QName"/>
</complexType>

<element name = "arithmeticVariable" type =
"expressionSchema:arithmeticVariableType"/>
<complexType name = "arithmeticVariableType">
  <attribute name = "avName" use = "required" type = "QName"/>
</complexType>

<element name = "arithmeticWithUnitVariable" type =
"expressionSchema:arithmeticWithUnitVariableType"/>
<complexType name = "arithmeticWithUnitVariableType">
  <attribute name = "aWUName" use = "required" type = "QName"/>
</complexType>

<element name = "stringVariable" type = "expressionSchema:stringVariableType"/>
<complexType name = "stringVariableType">
  <attribute name = "svName" use = "required" type = "QName"/>
</complexType>

<element name = "timeVariable" type = "expressionSchema:timeVariableType"/>
<complexType name = "timeVariableType">
  <attribute name = "tvName" use = "required" type = "QName"/>
</complexType>

<element name = "arrayVariable" type = "expressionSchema:arrayVariableType"/>
<complexType name = "arrayVariableType">
  <attribute name = "arrayVName" use = "required" type = "QName"/>
</complexType>

<element name = "stringConstant" type = "expressionSchema:stringConstantType"/>
<complexType name = "stringConstantType">

```

```
        <attribute name = "value" use = "required" type = "string"/>
    </complexType>
</schema>
```

## Appendix C: BNF notation for the grammar specified in the Expression

### Schema

---

booleanExpression ::= booleanConstant | booleanVariable | externalOperationResult |  
booleanExpression | quantifiedExpression | unaryBooleanOperator  
booleanExpression | booleanExpression binaryBooleanOperator  
booleanExpression | booleanExpression booleanComparator  
booleanExpression | arithmeticExpression arithmeticComparator  
arithmeticExpression | arithmeticWithUnitExpression  
arithmeticWithUnitComparator arithmeticWithUnitExpression |  
stringExpression stringComparator stringExpression | timeExpression  
timeComparator timeExpression

booleanConstant ::= 'true' | 'false'

unaryBooleanOperator ::= 'NOT'

binaryBooleanOperator ::= 'AND' | 'OR' | 'XOR' | 'EQUIVALENT' | 'IMPLIES'

quantifiedExpression ::= booleanExpression

booleanComparator ::= '==' | '!='

arithmeticExpression ::= arithmeticConstant | arithmeticVariable |

externalOperationResult | arithmeticExpression |

unaryArithmeticOperator arithmeticExpression | arithmeticExpression

binaryArithmeticOperator arithmeticExpression

arithmeticConstant ::= integerConstant | floatConstant | doubleConstant | longConstant

unaryArithmeticOperator ::= '-'

binaryArithmeticOperator ::= '+' | '-' | '\*' | '/' | '\*\*'

arithmeticComparator ::= '<' | '>' | '<=' | '>=' | '==' | '!='

arithmeticWithUnitExpression ::= numberWithUnitConstant |

arithmeticWithUnitVariable |

QoSmetric | arithmeticWithUnitExpression  
binaryArithmeticWithUnitOperator1 arithmeticWithUnitExpression  
| arithmeticWithUnitExpression  
binaryArithmeticWithUnitOperator2 arithmeticConstant

numberWithUnitConstant ::= number unit

arithmeticWithUnitComparator ::= '<' | '>' | '<=' | '>=' | '==' | '!='

binaryArithmeticWithUnitOperator1 ::= '+' | '-'

binaryArithmeticWithUnitOperator2 ::= '\*' | '/'

stringExpression ::= stringConstant | stringVariable | externalOperationResult |  
stringExpression

stringComparator ::= '<' | '>' | '<=' | '>=' | '==' | '!='

timeExpression ::= timeConstant | timeVariable | externalOperationResult |  
timeExpression | timeOperator timeExpression

timeConstant ::= time | date | date\_and\_time | time\_duration | gregorianYear |  
gregorianYearMonth | gregorianMonthDay | gregorianMonth | gregorianDay

timeOperator ::= 'BEFORE' | 'AFTER' | 'DURING' | 'BETWEEN'

timeComparator ::= '<' | '>' | '<=' | '>=' | '==' | '!='

## Appendix D: WSDL description for the buyStock Web Service example

---

```
<?xml version = "1.0" encoding = "UTF-8"?>

<wsdl:definitions targetNamespace =
"http://www.sce.carleton.ca/~kpatel/buyStock/buyStockService" name = "buyStockDefinition"
xmlns = "http://www.sce.carleton.ca/~kpatel/buyStock/buyStockService" xmlns:xsi =
"http://www.w3.org/2001/XMLSchema-instance" xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd = "http://www.w3.org/2001/XMLSchema" xmlns:xsd1 =
"http://www.sce.carleton.ca/~kpatel/buyStock/buyStockSchema" xmlns:tns =
"http://www.sce.carleton.ca/~kpatel/buyStock/buyStockService" xmlns:soap =
"http://schemas.xmlsoap.org/wsdl/soap/" xmlns:buyStock =
"http://www.sce.carleton.ca/~kpatel/buyStock/buyStockService" xsi:schemaLocation =
"http://schemas.xmlsoap.org/wsdl/
file:///D:/users/kpatel/WSOL_Parser/buyStockServiceData/WSDLSchema.xsd
http://schemas.xmlsoap.org/wsdl/soap/ SOAPSchema.xsd">

  <wsdl:import namespace =
"http://www.sce.carleton.ca/~kpatel/buyStock/buyStockSchema" location =
"file:///D:/users/kpatel/WSOL_Parser/buyStockServiceData/buyStockSchema.xsd"/>

  <wsdl:message name = "buySingleStockRequest">
    <wsdl:part name = "symbol" type = "xsd:string"/>
    <wsdl:part name = "quantity" type = "xsd:nonNegativeInteger"/>
    <wsdl:part name = "maxPrice" type = "xsd:float"/>
    <wsdl:part name = "buyBeforeDateTime" type = "xsd:dateTime"/>
  </wsdl:message>

  <wsdl:message name = "buyMultipleStockRequest">
    <wsdl:part name = "symbol_list" type = "xsd1:ArrayOfString"/>
    <wsdl:part name = "quantity_list" type = "xsd1:ArrayOfNonNegativeInteger"/>
    <wsdl:part name = "maxPrice_list" type = "xsd1:ArrayOfFloat"/>
    <wsdl:part name = "buyBeforeDateTime" type = "xsd:dateTime"/>
  </wsdl:message>

  <wsdl:message name = "buySingleStockResponse">
    <wsdl:part name = "stockOnSale" type = "xsd:boolean"/>
    <wsdl:part name = "stockMarketClosed" type = "xsd:boolean"/>
    <wsdl:part name = "totalPriceOfStockBought" type = "xsd:float"/>
    <wsdl:part name = "actualQuantityOfStockBought" type =
"xsd:nonNegativeInteger"/>
    <wsdl:part name = "actualPriceOfStockBought" type = "xsd:float"/>
    <wsdl:part name = "totalQuantityOfStockBought" type =
"xsd:nonNegativeInteger"/>
    <wsdl:part name = "totalStockBuyingCost" type = "xsd:float"/>
  </wsdl:message>

  <wsdl:message name = "buyMultipleStockResponse">
    <wsdl:part name = "stockOnSale_list" type = "xsd1:ArrayOfBoolean"/>
    <wsdl:part name = "stockMarketClosed" type = "xsd:boolean"/>
    <wsdl:part name = "totalPriceOfStockBought_list" type = "xsd1:ArrayOfFloat"/>
    <wsdl:part name = "actualQuantityOfStockBought_list" type =
"xsd1:ArrayOfNonNegativeInteger"/>
```

```

        <wsdl:part name = "actualPriceOfStockBought_list" type = "xsd1:ArrayOfFloat"/>
        <wsdl:part name = "totalQuantityOfStockBought" type =
        "xsd:nonNegativeInteger"/>
        <wsdl:part name = "totalStockBuyingCost" type = "xsd:float"/>
    </wsdl:message>

    <wsdl:message name = "buyStockFault"/>

    <wsdl:portType name = "buyStockPortType">
        <wsdl:operation name = "buySingleStockOperation">
            <wsdl:input name = "input" message = "tns:buySingleStockRequest"/>
            <wsdl:output name = "output" message =
            "tns:buySingleStockResponse"/>
            <wsdl:fault name = "fault" message = "tns:buyStockFault"/>
        </wsdl:operation>
        <wsdl:operation name = "buyMultipleStockOperation">
            <wsdl:input name = "input" message = "tns:buyMultipleStockRequest"/>
            <wsdl:output name = "output" message =
            "tns:buyMultipleStockResponse"/>
            <wsdl:fault name = "fault" message = "tns:buyStockFault"/>
        </wsdl:operation>
    </wsdl:portType>

    <wsdl:binding name = "buyStockBinding" type = "tns:buyStockPortType">
        <soap:binding style = "rpc" transport = "http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name = "buySingleStockOperation">
            <soap:operation soapAction = "urn:buyStock-service"/>
            <wsdl:input>
                <soap:body use = "encoded" encodingStyle =
                "http://schemas.xmlsoap.org/soap/encoding/" namespace =
                "urn:buyStock-service"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use = "encoded" encodingStyle =
                "http://schemas.xmlsoap.org/soap/encoding/" namespace =
                "urn:buyStock-service"/>
            </wsdl:output>
            <wsdl:fault name = "fault">
                <soap:body use = "encoded" encodingStyle =
                "http://schemas.xmlsoap.org/soap/encoding/" namespace =
                "urn:buyStock-service"/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name = "buyMultipleStockOperation">
            <soap:operation soapAction = "urn:buyStock-service"/>
            <wsdl:input>
                <soap:body use = "encoded" encodingStyle =
                "http://schemas.xmlsoap.org/soap/encoding/" namespace =
                "urn:buyStock-service"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use = "encoded" encodingStyle =
                "http://schemas.xmlsoap.org/soap/encoding/" namespace =
                "urn:buyStock-service"/>
            </wsdl:output>
            <wsdl:fault name = "fault">

```

```
        <soap:body use = "encoded" encodingStyle =  
            "http://schemas.xmlsoap.org/soap/encoding/" namespace =  
            "urn:buyStock-service"/>  
    </wsdl:fault>  
    </wsdl:operation>  
</wsdl:binding>  
  
<wsdl:service name = "buyStockService">  
    <wsdl:port name = "buyStockServicePort" binding = "tns:buyStockBinding">  
        <soap:address location = "http://localhost:8080/soap/servlet/rpcrouter"/>  
    </wsdl:port>  
</wsdl:service>  
  
</wsdl:definitions>
```



## Appendix E: WSOL description for the buyStock Web Service example

---

```
<?xml version = "1.0" encoding = "UTF-8"?>

<wsol:WSOLdefinitions ...>

  <wsol:externalOperationCall callID="extOp6"
    calledOperDescription="timeLookup:timeLookupPortType.currentTime"
    calledOperImplementation="timeLookup:timeLookupService.timeLookupServicePort"
    service="buyStock:buyStockService" portOrPortType="buyStock:buyStockServicePort"
    operation="WSOL-ANY"/>

  <wsol:constraint name = "C6" xsi:type = "preConditionSchema:preCondition" service =
    "buyStock:buyStockService" portOrPortType = "buyStock:buyStockServicePort"
    operation = "WSOL-ANY">
    <expressionSchema:booleanExpression>
      <expressionSchema:booleanExpression>
        <expressionSchema:timeExpression>
          <expressionSchema:externalOperationResult callID =
            "tns:extOp6" resultPartName =
            "timeService:currentTimeResponse.currentTime"/>
          </expressionSchema:timeExpression>
          <expressionSchema:timeComparator type = "==" />
          <expressionSchema:timeExpression>
            <expressionSchema:timeOperator type = "AFTER" />
            <expressionSchema:timeExpression>
              <expressionSchema:timeConstant>
                <expressionSchema:time value =
                  "09:00:00-05:00" />
              </expressionSchema:timeConstant>
            </expressionSchema:timeExpression>
          </expressionSchema:timeExpression>
        </expressionSchema:booleanExpression>
        <expressionSchema:binaryBooleanOperator type = "AND" />
        <expressionSchema:booleanExpression>
          <expressionSchema:timeExpression>
            <expressionSchema:externalOperationResult callID =
              "tns:extOp6" resultPartName = "timeService:currentTimeResponse.currentTime" />
            </expressionSchema:timeExpression>
            <expressionSchema:timeComparator type = "==" />
            <expressionSchema:timeExpression>
              <expressionSchema:timeOperator type = "BEFORE" />
              <expressionSchema:timeExpression>
                <expressionSchema:timeConstant>
                  <expressionSchema:time value =
                    "17:00:00-05:00" />
                </expressionSchema:timeConstant>
              </expressionSchema:timeExpression>
            </expressionSchema:timeExpression>
          </expressionSchema:booleanExpression>
        </expressionSchema:booleanExpression>
      </wsol:constraint>
```

```

<wsol:CGT name = "CGT2" service = "buyStock:buyStockService" portOrPortType =
"buyStock:buyStockServicePort" operation = "buyStock:buySingleStockOperation">

  <wsol:parameter name = "responseTime" dataType = "wsol:numberWithUnit"
unit = "QoSMeasOntology:millisecond"/>

  <wsol:constraint name = "C50" xsi:type = "qosSchema:QoSconstraint" service =
"buyStock:buyStockService" portOrPortType = "buyStock:buyStockServicePort"
operation = "buyStock:buySingleStockOperation">
    <expressionSchema:booleanExpression>
        <expressionSchema:arithmeticWithUnitExpression>
            <expressionSchema:QoSmetric metricType =
"QoSMetricOntology:ResponseTime" metricUnit =
"QoSMeasOntology:millisecond" service =
"buyStock:buyStockService" portOrPortType =
"buyStock:buyStockServicePort" operation =
"buyStock:buySingleStockOperation" measuredBy =
"WSOL_INTERNAL"/>
        </expressionSchema:arithmeticWithUnitExpression>
        <expressionSchema:arithmeticWithUnitComparator type =
"&lt;"/>
        <expressionSchema:arithmeticWithUnitExpression>
            <expressionSchema:arithmeticWithUnitVariable
aWUName = "tns:CGT2.responseTime"/>
        </expressionSchema:arithmeticWithUnitExpression>
    </expressionSchema:booleanExpression>
</wsol:constraint>

</wsol:CGT>

<wsol:CGT name = "CGT3" extends = "tns:CGT2" service = "buyStock:buyStockService"
portOrPortType = "buyStock:buyStockServicePort" operation =
"buyStock:buySingleStockOperation">

  <wsol:constraint name = "C60" xsi:type = "qosSchema:QoSconstraint" service =
"buyStock:buyStockService" portOrPortType = "buyStock:buyStockServicePort"
operation = "buyStock:buySingleStockOperation">
    <expressionSchema:booleanExpression>
        <expressionSchema:arithmeticWithUnitExpression>
            <expressionSchema:QoSmetric metricType =
"QoSMetricOntology:ResponseTime" metricUnit =
"QoSMeasOntology:millisecond" service =
"buyStock:buyStockService" portOrPortType =
"buyStock:buyStockServicePort" operation =
"buyStock:buySingleStockOperation" measuredBy =
"WSOL_INTERNAL"/>
        </expressionSchema:arithmeticWithUnitExpression>
        <expressionSchema:arithmeticWithUnitComparator type =
"&lt;"/>
        <expressionSchema:arithmeticWithUnitExpression>
            <expressionSchema:arithmeticWithUnitVariable
aWUName = "tns:CGT3.responseTime"/>
        </expressionSchema:arithmeticWithUnitExpression>
    </expressionSchema:booleanExpression>
</wsol:constraint>

```

</wsol:CGT>

<wsol:serviceOffering name = "SO1" service = "buyStock:buyStockService">

<wsol:externalOperationCall callID="extOp1"  
calledOperDescription="symbolLookup:symbolLookupPortType.symbolExists"  
calledOperImplementation="symbolLookup:symbolLookupService.symbolLookup  
ServicePort" service="buyStock:buyStockService"  
portOrPortType="buyStock:buyStockServicePort"  
operation="buyStock:buySingleStockOperation">

<wsol:callList>

<wsol:parameterValue>

<wsol:partName

name="symbolLookup:singleSymbolLookupRequest.symbol"/>

<wsol:paramValue>

<expressionSchema:stringExpression>

<expressionSchema:stringVariable

svName="buyStock:buySingleStockRequest.symbol"/>

</expressionSchema:stringExpression>

</wsol:paramValue>

</wsol:parameterValue>

</wsol:callList>

</wsol:externalOperationCall>

<wsol:constraint name = "C5" xsi:type = "preConditionSchema:preCondition"  
service = "buyStock:buyStockService" portOrPortType =  
"buyStock:buyStockServicePort" operation =  
"buyStock:buySingleStockOperation">

<expressionSchema:booleanExpression>

<expressionSchema:booleanExpression>

<expressionSchema:externalOperationResult callID =

"tns:SO1.extOp1" resultPartName =

"symbolLookup:symbolLookupResponse.symbol\_exists"  
/>

</expressionSchema:booleanExpression>

<expressionSchema:booleanComparator type = "==">

<expressionSchema:booleanExpression>

<expressionSchema:booleanConstant type = "true"/>

</expressionSchema:booleanExpression>

</expressionSchema:booleanExpression>

</wsol:constraint>

<wsol:constraint name = "C3" xsi:type = "preConditionSchema:preCondition"  
service = "buyStock:buyStockService" portOrPortType =  
"buyStock:buyStockServicePort" operation =  
"buyStock:buySingleStockOperation">

<expressionSchema:booleanExpression>

<expressionSchema:arithmeticExpression>

<expressionSchema:arithmeticVariable avName =

"buyStock:buySingleStockRequest.quantity"/>

</expressionSchema:arithmeticExpression>

<expressionSchema:arithmeticComparator type = ">">

<expressionSchema:arithmeticExpression>

<expressionSchema:arithmeticConstant>

```

                <expressionSchema:integerConstant value =
                    "0"/>
            </expressionSchema:arithmeticConstant>
        </expressionSchema:arithmeticExpression>
    </expressionSchema:booleanExpression>
</wsol:constraint>

<wsol:constraint name = "AccRghtCons2" xsi:type =
"accessRightSchema:accessRight" service = "buyStock:buyStockService"
portOrPortType = "buyStock:buyStockServicePort" operation =
"buyStock:buyMultipleStockOperation">
    <expressionSchema:booleanExpression>
        <expressionSchema:booleanConstant type = "true"/>
    </expressionSchema:booleanExpression>
</wsol:constraint>

<wsol:constraint name = "QoScons1" xsi:type = "qosSchema:QoSconstraint"
service = "buyStock:buyStockService" portOrPortType =
"buyStock:buyStockServicePort" operation =
"buyStock:buySingleStockOperation">
    <expressionSchema:booleanExpression>
        <expressionSchema:arithmeticWithUnitExpression>
            <expressionSchema:QoSmetric metricType =
                "QoSMetricOntology:ResponseTime" metricUnit =
                "QoSMeasOntology:millisecond" service =
                "buyStock:buyStockService" portOrPortType =
                "buyStock:buyStockServicePort" operation =
                "buyStock:buySingleStockOperation" measuredBy =
                "WSOL_INTERNAL"/>
            </expressionSchema:arithmeticWithUnitExpression>
            <expressionSchema:arithmeticWithUnitComparator type
                = "&lt;"/>
            <expressionSchema:arithmeticWithUnitExpression>
                <wsol:numberWithUnitConstant>
                    <wsol:number value = "10"/>
                    <wsol:unit type =
                        "QoSMeasOntology:millisecond"/>
                </wsol:numberWithUnitConstant>
            </expressionSchema:arithmeticWithUnitExpression>
        </expressionSchema:booleanExpression>
    </wsol:constraint>

<wsol:subscription>
    <wsol:numberWithUnitConstant>
        <wsol:number value = "5"/>
        <wsol:unit type = "currencyOntology:Dollar"/>
    </wsol:numberWithUnitConstant>
    <wsol:subscriptionDuration duration = "P1Y"/>
</wsol:subscription>

<wsol:price name = "Price1" service = "buyStock:buyStockService"
portOrPortType = "buyStock:buyStockServicePort" operation =
"buyStock:buySingleStockOperation">
    <wsol:numberWithUnitConstant>
        <wsol:number value = "5"/>
        <wsol:unit type = "currencyOntology:Dollar"/>
    </wsol:numberWithUnitConstant>
</wsol:price>

```

```

        </wsol:numberWithUnitConstant>
</wsol:price>

<wsol:priceDefault>
    <wsol:numberWithUnitConstant>
        <wsol:number value = "2"/>
        <wsol:unit type = "currencyOntology:Dollar"/>
    </wsol:numberWithUnitConstant>
</wsol:priceDefault>

<wsol:penalty name = "Penalty1" service = "buyStock:buyStockService"
portOrPortType = "buyStock:buyStockServicePort" operation =
"buyStock:buySingleStockOperation">
    <wsol:numberWithUnitConstant>
        <wsol:number value = "-2.5"/>
        <wsol:unit type = "currencyOntology:Dollar"/>
    </wsol:numberWithUnitConstant>
</wsol:penalty>

<wsol:penaltyDefault>
    <wsol:numberWithUnitConstant>
        <wsol:number value = "-1"/>
        <wsol:unit type = "currencyOntology:Dollar"/>
    </wsol:numberWithUnitConstant>
</wsol:penaltyDefault>

<wsol:managementResponsibility name = "MangRespSO1">
    <wsol:supplierResponsibility scope = "tns:SO1.AccRghtCons2"/>
    <wsol:consumerResponsibility scope = "tns:SO1.C3"/>
</wsol:managementResponsibility>

<wsol:CG name = "CG6" service = "buyStock:buyStockService" portOrPortType
= "buyStock:buyStockServicePort" operation =
"buyStock:buySingleStockOperation">

    <wsol:constraint name = "C4" xsi:type =
"preConditionSchema:preCondition" service =
"buyStock:buyStockService" portOrPortType =
"buyStock:buyStockServicePort" operation =
"buyStock:buySingleStockOperation">
        <expressionSchema:booleanExpression>
            <expressionSchema:arithmeticExpression>
                <expressionSchema:arithmeticVariable avName
=
                "buyStock:buySingleStockRequest.maxPrice"/>
            </expressionSchema:arithmeticExpression>
            <expressionSchema:arithmeticComparator type = ">"/>
            <expressionSchema:arithmeticExpression>
                <expressionSchema:arithmeticConstant>
                    <expressionSchema:floatConstant value
= "0.0"/>
                </expressionSchema:arithmeticConstant>
            </expressionSchema:arithmeticExpression>
        </expressionSchema:booleanExpression>
    </wsol:constraint>

```

```

        <wsol:include constructName = "tns:SO1.C3" resService =
        "buyStock:buyStockService" resPortOrPortType =
        "buyStock:buyStockServicePort" resOperation =
        "buyStock:buySingleStockOperation" resName = "NewC3"/>

    </wsol:CG>

    <wsol:instantiate CGTName = "CGT2" resService = "buyStock:buyStockService"
    resPortOrPortType = "buyStock:buyStockServicePort" resOperation =
    "buyStock:buySingleStockOperation" resCGName = "CG5">
        <wsol:parmValue name = "responseTime">
            <wsol:numberWithUnitConstant>
                <wsol:value>30</wsol:value>
                <wsol:unit type = "QoSMeasOntology:millisecond"/>
            </wsol:numberWithUnitConstant>
        </wsol:parmValue>
    </wsol:instantiate>

</wsol:serviceOffering>

<wsol:serviceOffering name = "SO2" extends = "tns:SO1" service =
"buyStock:buyStockService">

    <wsol:constraint name = "C9" xsi:type = "preConditionSchema:preCondition"
    service = "buyStock:buyStockService" portOrPortType =
    "buyStock:buyStockServicePort" portType = "buyStock:buyStockPortType"
    operation = "buyStock:buyMultipleStockOperation">
        <expressionSchema:booleanExpression>
            <expressionSchema:quantifiedExpression type = "FOR_ALL_IN"
            arrayVariableName =
            "buyStock:buyMultipleStockRequest.limitPrice_list">
                <expressionSchema:booleanExpression>
                    <expressionSchema:arithmeticExpression>
                        <expressionSchema:arithmeticVariable
                        avName =
                        "buyStock:buyMultipleStockRequest.limi
                        tPrice_list.WSOLARRAYEL"/>
                    </expressionSchema:arithmeticExpression>
                    <expressionSchema:arithmeticComparator type
                    = ">"/>
                    <expressionSchema:arithmeticExpression>
                        <expressionSchema:arithmeticConstant>
                            <expressionSchema:floatConstant value
                            = "0.0"/>
                        </expressionSchema:arithmeticConstant>
                    </expressionSchema:arithmeticExpression>
                </expressionSchema:booleanExpression>
            </expressionSchema:quantifiedExpression>
        </expressionSchema:booleanExpression>
    </wsol:constraint>

</wsol:serviceOffering>

<wsol:serviceOffering name = "SO3" service = "buyStock:buyStockService"
accountingParty = "http://www.thirdParty.com">

```

```

<wsol:externalOperationCall callID="extOp3"
calledOperDescription="mathLibrary:mathLibraryPortType.sumOfFloatArrayElem
ents" calledOperImplementation="WSOL-INTERNAL"
service="buyStock:buyStockService"
portOrPortType="buyStock:buyStockServicePort"
operation="buyStock:buyMultipleStockOperation">
  <wsol:callList>
    <wsol:parameterValue>
      <wsol:partName
name="mathLibrary:floatArraySumRequest.array"/>
      <wsol:paramValue>
        <expressionSchema:arithmeticExpression>
          <expressionSchema:arrayVariable
arrayVName="buyStock:buyMultipleSto
ckResponse.totalPriceOfStockBought_li
st"/>
        </expressionSchema:arithmeticExpression>
      </wsol:paramValue>
    </wsol:parameterValue>
  </wsol:callList>
</wsol:externalOperationCall>

```

```

<wsol:constraint name = "C27" xsi:type = "postConditionSchema:postCondition"
service = "buyStock:buyStockService" portOrPortType =
"buyStock:buyStockServicePort" operation =
"buyStock:buyMultipleStockOperation">
  <expressionSchema:booleanExpression>
    <expressionSchema:arithmeticExpression>
      <expressionSchema:arithmeticVariable avName =
"buyStock:buyMultipleStockResponse.totalStockBuying
Cost"/>
    </expressionSchema:arithmeticExpression>
    <expressionSchema:arithmeticComparator type = "==" />
    <expressionSchema:arithmeticExpression>
      <expressionSchema:externalOperationResult callID =
"tns:SO3.extOp3" resultPartName =
"mathLibrary:floatArraySumResponse.arraySum"/>
    </expressionSchema:arithmeticExpression>
  </expressionSchema:booleanExpression>
</wsol:constraint>
<wsol:managementResponsibility name = "MangRespSO3">
  <wsol:independentResponsibility scope = "tns:SO3" entity =
"http://www.thirdParty.com"/>
</wsol:managementResponsibility>
</wsol:serviceOffering>

```

```

<wsol:serviceOffering name = "SO4" service = "buyStock:buyStockService"
accountingParty = "WSOL-CONSUMERWS">

```

```

  <wsol:CG name = "CG10" extends = "tns:SO1.CG6" service =
"buyStock:buyStockService" portOrPortType = "buyStock:buyStockServicePort"
operation = "buyStock:buySingleStockOperation"/>

```

```

</wsol:serviceOffering>

```

```

</wsol:WSOLdefinitions>

```

## Appendix F: “QoSMeasOntology.ont” (An Ontology of QoS measurement units)

---

```
<?xml version="1.0" encoding="UTF-8"?>

<QoSMeasOntSchema:QoSMeasDefinitions
xmlns:QoSMeasOntSchema="http://www.sce.carleton.ca/~kpatel/WSOL/QoSMeasOntSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.sce.carleton.ca/~kpatel/WSOL/QoSMeasOntSchema
QoSMeasOntSchema.xsd"
xmlns:tns="http://www.sce.carleton.ca/~kpatel/WSOL/QoSMeasOntology"
xmlns="http://www.sce.carleton.ca/~kpatel/WSOL/QoSMeasOntology"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:QoSMeasOntology="http://www.sce.carleton.ca/~kpatel/WSOL/QoSMeasOntology"
targetNamespace="http://www.sce.carleton.ca/~kpatel/WSOL/QoSMeasOntology">

    <QoSMeasOntSchema:QoSMeasUnit name="millisecond"/>

    <QoSMeasOntSchema:QoSMeasUnit name="second"/>

    <QoSMeasOntSchema:QoSMeasUnit name="minute"/>

    <QoSMeasOntSchema:QoSMeasUnit name="hour"/>

</QoSMeasOntSchema:QoSMeasDefinitions>
```