

Charles University in Prague
Faculty of Mathematics and Physics

Master Thesis



Tomáš Knap

Comparison of Fully Software and Hardware Accelerated XML Processing

Department of Software Engineering
Supervisor: *RNDr. Irena Mlýnková, Ph.D.*
Study plan: *Informatics*

I would like to thank to my supervisor Dr. Irena Mlýnková for many inspiring ideas and advices, provided sets of possible testing data, overview of benchmarks, and principally for her conscionable and valuable leadership in all respects. Furthermore, I would like to thank to Dr. Jakub Yaghob for smooth installation and maintenance of the servers for the testing environment and valuable ideas especially at the beginning of the work.

Moreover, I would certainly like to thank to Jiří Melichna, former IBM WebSphere DataPower Technical Sale for CEMAAS and currently SOA Architect, for its selfless support in all phases of the work, valuable hints during the preparation of the tests, many ideas which accelerates my work, and dozens of pleasant testing afternoons with interesting discussions. I hope I will be able to reciprocate in the future. Besides, I would like to thank to Miroslav Vaic and Tomáš Peroutka from the Trask solutions company for initial ideas and procurement of access to the XML processing appliance.

Finally, special thanks to my parents, who supported me throughout the writing, and to my friends, who provided me valuable relaxation in my spare time.

I hereby declare that I have written this work independently and used no resources other than the indicated aids. I agree with lending of this work. The work may be reproduced for academic purposes.

Prague, 30th July 2008

Tomáš Knap

Table of Contents

I. Introduction and Basic Terminology.....	10
1. Introduction	10
1.1. Background Motivation.....	10
1.2. Goals of The Thesis.....	10
1.3. Conventions.....	10
1.3.1. Hierarchy of Headings	11
1.4. Thesis Organization.....	11
2. Terminology and Technology Overview.....	12
2.1. XML.....	12
2.1.1. Parsing XML Documents	12
2.1.2. Validating XML Documents	13
2.1.3. Transforming XML Documents	18
2.1.4. Securing XML Documents	20
2.2. XML in Java.....	25
2.2.1. Java API for XML Processing	25
2.2.2. Java API for XML Digital Signature and XML Encryption.....	27
2.3. J2EE and Servlets.....	27
II. Testing Environments.....	28
3. Description of Testing Environments.....	28
3.1. IBM WebSphere DataPower Integration Appliance XI50.....	28
3.1.1. Features of XI50	28
3.1.2. Typical Use Cases of XI50	30
3.1.3. Connection to a Business Network	31
3.1.4. Web-based GUI	32
3.1.5. Internal Device Structure	33
3.1.6. Approximate Price	34
3.1.7. Other Appliances	34
3.2. IBM WebSphere Application Server v 6.1	34
3.2.1. Architectural Overview.....	34
3.2.2. Installation and Profiling	35
3.2.3. Administration of the Application Server.....	35
3.2.4. Approximate Price	36
3.2.5. Other Application Servers or SW Solutions	36
3.3. Auxiliary Tools	36
3.3.1. Eclipse Version 3	37
3.3.2. IBM Rational Application Developer Version 7	37
3.3.3. Curl	37
3.3.4. Apache HTTP Server Benchmark Tool (AB).....	37
3.3.5. Windows Server 2003 Performance Tool.....	39
III. Testing - Defining Models and Metrics	40
4. Overview of the Testing Suites	40
4.1. Naming Conventions.....	40
4.2. Process of Searching Suitable Collections of Testing Scenarios	40
4.2.1. Way to the “Onion” Testing Suite	40
4.2.2. Way to the “Flat” Testing Suite	41

4.3.	Outline of the Testing Hierarchy.....	42
4.4.	Architecture of the Testing Framework	43
4.5.	Server Part of the Testing Framework in HW Environment.....	44
4.5.1.	Front End and Back End Sections	45
4.5.2.	Firewall Type	46
4.5.3.	XML Manager	46
4.5.4.	Firewall Policy	46
4.5.5.	XML Firewall Used in Our Testing Framework	49
4.6.	Server Part of the Testing Framework in the SW Environment	50
4.6.1.	Processing Rules	51
4.7.	Measuring the Tests	55
4.7.1.	Measures	56
5.	Defining the “Flat” Testing Suite	57
5.1.	The Outline of the Testing Groups and Testing Scenarios	57
5.2.	Testing Group: Parsing XML Data	57
5.2.1.	Testing Data and Testing Hierarchy	57
5.2.2.	Processing Rule Definition	59
5.2.1.	Tested Engines in the SW Environment	61
5.3.	Testing Group: Validating XML Data	61
5.3.1.	Testing Data and Testing Hierarchy	61
5.3.1.	Processing Rule Definition	62
5.3.1.	Tested Engines in the SW Environment	63
5.4.	Testing Group: Transforming XML Data	63
5.4.1.	Testing Data	63
5.4.1.	Processing Rule Definition	68
5.4.2.	Tested Engines in the SW Environment	69
5.5.	Testing Group: Securing XML Data	70
5.5.1.	Testing Data and Testing Hierarchy	70
5.5.2.	Processing Rule Definition	70
5.5.3.	Tested Engines in the SW Environment	74
6.	Defining the “Onion” Testing Suite	76
6.1.	The Outline of the Testing Groups and Testing Scenarios	76
6.2.	Testing Group: Auction.....	76
6.2.1.	Testing Data and Testing Hierarchy	76
6.2.2.	Processing Rule Definition (Auction_XSLT).....	77
6.2.3.	Processing Rule Definition (Auction_VAL_XSLT)	77
6.2.4.	Processing Rule Definition (Auction_VAL_XSLT_SIGN)	78
6.2.5.	Processing Rule Definition (Auction_VAL_XSLT_SIGN_ENC)	79
6.2.6.	Tested Engines in the SW Environment	80
6.3.	Testing Group: CSVOutput.....	80
6.3.1.	Testing Data and Testing Hierarchy	81
6.3.2.	Processing Rule Definition (CSV_XSLT).....	82
6.3.3.	Processing Rule Definition (CSV_XSLT_XSLT2).....	82
6.3.4.	Processing Rule Definition (CSV_XSLT_VAL_XSLT2).....	82
6.3.5.	Processing Rule Definition (CSV_VER_XSLT3_XSLT_VAL_XSLT2)	83
6.3.6.	Rule Definition (CSV_DEC_VER_XSLT3_XSLT_VAL_XSLT2).....	84
6.3.7.	Tested Engines in the SW Environment	85
IV. Testing - Testing and Comparing		86
7.	Environment Settings	86

7.1. The HW Environment	86
7.1.1. Configuration	86
7.1.2. Tuning Possibilities.....	86
7.1.3. Monitoring Capabilities	87
7.1.4. Debugging Capabilities.....	87
7.2. The SW Environment.....	88
7.2.1. Configuration	88
7.2.2. Tuning Possibilities.....	89
7.2.3. Monitoring and Debugging Capabilities.....	93
7.3. Used Settings.....	93
8. Testing	94
8.1. The “Flat” Testing Suite.....	95
8.1.1. Parsing Testing Group	95
8.1.2. Validating Testing Group	97
8.1.3. Transforming Testing Group	99
8.1.4. Securing Testing Group	102
8.1.5. Cross-scenario Comparison	106
8.2. The “Onion” Testing Suite.....	109
8.2.1. Auction Testing Group	110
8.2.2. CSVOutput Testing Group	113
8.2.1. Cross-scenario Comparison	117
V. Results, Conclusion, and Future Work	120
9. Conclusion.....	120
10. Future Work.....	122
VI. References and Appendices	123
11. References	123
12. Appendices	128
12.1. Appendix A - Contents of the Enclosed DVD-ROM.....	128
12.2. Appendix B – Important Java Packages.....	128
12.2.1. Java API for XML Processing 1.3 - JSR 206 [W26].....	128
12.2.2. XML Digital Signature API for Java 1.0 - JSR 105 [W20].....	129
12.2.3. XML Digital Encryption API for Java 1.0 - JSR 106 [W21]	129

List of Figures and Examples

Figure 1	Conventions.....	11
Figure 2	Sample XML Document	12
Figure 3	XML Document for Explanation of DTD Validation (peopleList.xml)	14
Figure 4	Sample DTD File (peopleList.dtd).....	14
Figure 5	Sample XML Document for Explanation of XSD Validation (peopleList.xml)	15
Figure 6	Sample XSD File (peopleList.xsd).....	16
Figure 7	Source XML Document for Explanation of XSL Transformations	19
Figure 8	XSL Stylesheet for Creating the Resulting HTML Page	19
Figure 9	Resulting XML Document for Explanation of XSL Transformations.....	20
Figure 10	Process of Ensuring Integrity	21
Figure 11	Process of Ensuring Confidentiality.....	21
Figure 12	Spheres of Activity of a Security Context.....	22
Figure 13	Example of Using XML Signature.....	23
Figure 14	XML Document for Explanation of Encrypting	24
Figure 15	XML Document with Encrypted Credit Card Number.....	24
Figure 16	Information about a Key Used for Encryption and Decryption	25
Figure 17	SAX Parsing Example.....	26
Figure 18	DOM Parsing Example	26
Figure 19	XSL Transformation Example	27
Figure 20	XI50 – Outer Face of the Appliance	28
Figure 21	Example of a Flat File	31
Figure 22	Flat File Description (FFD) File for the Above Flat File.....	31
Figure 23	Proxy Mode of XI50 Connection to a Network	32
Figure 24	Coprocessor Mode of XI50 Connection to a Network.....	32
Figure 25	Web-based GUI - Main Window	33
Figure 26	Simple Architectural Overview of Application Server	35
Figure 27	Administrative Console for IBM WebSphere Application Server.....	36
Figure 28	Curl Sample Request.....	37
Figure 29	AB Sample Request	38
Figure 30	AB Sample Response	38
Figure 31	Sample Processing Rule of a Testing Scenario in the “Onion” Testing Suite	41
Figure 32	Architecture of the Testing Framework in the SW Environment	43
Figure 33	Architecture of the Testing Framework in the HW Environment.....	44
Figure 34	Configuring XML Firewall	45
Figure 35	Sample Processing Policy with a Set of Processing Rules.....	47
Figure 36	Common Client to Server Processing Rule in HW Environment	49
Figure 37	Detail of Matching Action.....	50
Figure 38	Detail of Results Action	50
Figure 39	Sequence Diagram of Processing Individual and Pipelined Actions	51
Figure 40	A Processing Rule with a Transforming Action	54
Figure 41	A Processing Rule with an Encrypting Action and TreeWrapper Actions	54
Figure 42	A Complex Rule with DOM and PIPED Input and Output Contexts.....	54
Figure 43	Definition of a Processing Rule	55
Figure 44	Names and Appropriate Sizes of Used XMark Documents.....	58
Figure 45	Processing Rule for Par_STREAM Testing Scenario.....	59
Figure 46	Processing Rule for Par_DOM Testing Scenario.....	59
Figure 47	Processing Rule for Stream Parsing	60
Figure 48	Detail of the Transforming Action	60

Figure 49	Processing Rule for Val_XSD_BASE Testing Scenario	62
Figure 50	Processing Rule for Validating Action.....	62
Figure 51	Detail of Validating Action	63
Figure 52	Processing Rule for XSLTMark Test Cases in SW	68
Figure 53	Processing Rule for XSLTMark Test Cases in HW	69
Figure 54	Detail of Transforming Action.....	69
Figure 55	Signing Processing Rule for DSA and SHA1 Algorithms.....	71
Figure 56	Verifying Processing Rule for DSA and SHA1 Algorithms.....	71
Figure 57	Encrypting using RSA2 and AES 256	71
Figure 58	Detail of Encrypting Using RSA2 and AES 256	72
Figure 59	Signing Using DSA and SHA1 Algorithms.....	73
Figure 60	Detail of Signing Using DSA and SHA1 Algorithms.....	74
Figure 61	Names and Appropriate Sizes of Used XMark Documents in Auction	76
Figure 62	Processing Rule for Auction_XSLT Testing Scenario	77
Figure 63	Processing Rule for Auction_VAL_XSLT Testing Scenario in SW	78
Figure 64	Processing Rule for Auction_VAL_XSLT Testing Scenario in HW	78
Figure 65	Processing Rule for Auction_VAL_XSLT_SIGN Testing Scenario in SW	79
Figure 66	Processing Rule for Auction_VAL_XSLT_SIGN Testing Scenario in HW	79
Figure 67	Rule for Auction_VAL_XSLT_SIGN_ENC Testing Scenario in SW	80
Figure 68	Rule for Auction_VAL_XSLT_SIGN_ENC Testing Scenario in HW	80
Figure 69	Names and Appropriate Sizes of used XML Documents in CSVOutput	81
Figure 70	Structure of the Database of Employees	81
Figure 71	Processing Rule for CSV_XSLT Testing Scenario.....	82
Figure 72	Processing Rule for CSV_XSLT_XSLT2 Testing Scenario.....	82
Figure 73	Processing Rule for CSV_XSLT_VAL_XSLT2 Testing Scenario	83
Figure 74	Rule for CSV_VER_XSLT3_XSLT_VAL_XSLT2 Testing Scenario in SW....	83
Figure 75	Rule for CSV_VER_XSLT3_XSLT_VAL_XSLT2 Testing Scenario in HW ..	84
Figure 76	Rule for CSV_DEC_VER_XSLT3_XSLT_VAL_XSLT2 Scenario in SW.....	84
Figure 77	Rule for CSV_DEC_VER_XSLT3_XSLT_VAL_XSLT2 Scenario in HW	85
Figure 78	CPU Usage (XI50)	87
Figure 79	System Usage (XI50)	87
Figure 80	Memory Usage (XI50)	87
Figure 81	System Log (XI50).....	87
Figure 82	Probe - Selecting Context and Viewing XML Document.....	88
Figure 83	Probe - Context Variables	88
Figure 84	Number of GC Runs.....	91
Figure 85	Duration of One GC Run	91
Figure 86	Percentage of Time Spent in GC.....	91
Figure 87	Requests per Second Measure.....	92
Figure 88	Used Settings.....	93
Figure 89	DOM and Stream Testing Scenarios in HW and SW	96
Figure 90	Throughput Ratio of DOM Parsing.....	97
Figure 91	Validating Testing Group in HW and SW	98
Figure 92	XSLTMark in HW and SW.....	99
Figure 93	XALAN (on the Left) and SAXON-B (on the Right) Heap Usage	100
Figure 94	Comparison of Throughputs of XSLTMark_XL	101
Figure 95	Comparison of Throughputs of Transforming Testing Group in HW	102
Figure 96	Encrypting in HW and SW.....	103
Figure 97	Signing in HW and SW	104
Figure 98	Comparison of Signing and Encrypting in HW and SW	105

Figure 99	Summary Comparison of Throughputs in the HW Environment	106
Figure 100	Summary Comparison of Throughputs in the SW Environment	107
Figure 101	Network Utilization of the “Flat” Testing Suite with Optimal Test Cases	108
Figure 102	Network Utilization of the “Flat” Testing Suite in HW	108
Figure 103	Network Utilization of the “Flat” Testing Suite in SW	109
Figure 104	Cross-scenario Comparison of Test0.002 in Both Environments	110
Figure 105	Cross-scenario Comparison of Test0.002 in the SW Environment	111
Figure 106	Cross-size comparison of Test Cases of Auction_XSLT Testing Scenario	112
Figure 107	Cross-Scenario and Cross-size Comparison for C=1 in HW and SW	113
Figure 108	Cross-scenario Throughput for Rows200 Test Case in HW	114
Figure 109	Cross-scenario Throughput for Rows200 Test Case in SW	115
Figure 110	Cross-scenario Throughput for Rows2000 Test Case in SW	115
Figure 111	Cross-size Throughput of CSV_XSLT Testing Scenario in HW and SW	116
Figure 112	Cross-Scenario and Cross-size Comparison for C=1 in HW and SW	117
Figure 113	Network Utilization of the “Onion” Testing Suite with Optimal Test Cases ...	118
Figure 114	Network Utilization of the “Onion” Testing Suite in HW	118
Figure 115	Network Utilization of the “Onion” Testing Suite in SW	119
Figure 116	Packages of Java API for XML Processing	129
Figure 117	Packages of XML Digital Signature API for Java	129
Figure 118	Packages of XML Digital Encryption API for Java	129

Title: *Comparison of Fully Software and Hardware Accelerated XML Processing*

Author: *Tomáš Knap*

Department: *Department of Software Engineering*

Supervisor: *RNDr. Irena Mlýnková, Ph.D.*

Supervisor's e-mail address: *Irena.Mlynkova@mff.cuni.cz*

Abstract:

The aim of this work is to compare XML processing abilities of standard software solutions and hardware accelerated scenarios using a new generation of XML processing appliances. The emphasis is puts on the speed of processing XML documents and on the demandingness of various operations over XML data. Firstly, we describe the used XML technologies and corresponding implementations in Java. Consequently, we characterize the core parts of our testing frameworks - IBM WebSphere DataPower Integration Appliance XI50 for hardware accelerated and IBM WebSphere Application Server 6.1 for standard XML processing. Further, the testing hierarchy involving two distinct testing suites - "Flat" and "Onion" - and tens of testing scenarios are defined. The "Flat" testing suite covers parsing, validating, transforming, and securing operations over XML data applied individually to a wide range of testing data, without bothering with concurrency. On the other hand, the "Onion" testing suite is a stress test combining several operations together. Both testing suites are executed on our testing framework and several measures (such as throughput) are collected and analyzed using n-dimensional OLAP cubes. The results show under which circumstances the appliance for hardware accelerated XML processing is worth using on and quantify the gain, which can be reached when incorporating such appliance to a network.

Keywords: *XML, XSLT, XML Security, XML Benchmark, XML Firewall*

Název práce: *Srovnání softwarového a hardwarově akcelerovaného zpracování XML dat*

Autor: *Tomáš Knap*

Katedra (ústav): *Katedra softwarového inženýrství*

Vedoucí diplomové práce: *RNDr. Irena Mlýnková, Ph.D.*

E-mail vedoucího: *Irena.Mlynkova@mff.cuni.cz*

Abstrakt:

Cílem diplomové práce je porovnat možnosti zpracování XML dokumentů pomocí standardních softwarových řešení a s využitím speciálních zařízení pro hardwarovou akceleraci zpracovávaných XML dokumentů. V první části jsou popsány použité XML technologie, včetně jejich implementací v Javě. Následně jsou představeny klíčové části testovaných systémů - IBM WebSphere DataPower Integration Appliance XI50 pro hardwarově akcelerované a IBM WebSphere Application Server 6.1 pro standardní softwarové zpracování XML dat. Následně jsou definovány desítky testovacích scénářů, které můžeme rozřadit do dvou hlavních skupin - „Flat“ a „Onion“. V prvně jmenované skupině jsou individuálně otestovány standardní operace nad XML daty jako parsování, validace, transformace a šifrování. V „Onion“ testovací skupině jsou pak zátěžové testy kombinující více operací nad XML daty. Výsledky testů obou skupin jsou posbírány a analyzovány v OLAP kostce. Výsledky ukazují, kdy se zařízení podporující hardwarově akcelerované zpracování XML dat vyplatí a také kvantifikují zvýšení propustnosti dat v podnikové síti po začlenění tohoto zařízení.

Klíčová slova: *XML, XSLT, Zabezpečení XML, XML Benchmark, XML Firewall*

I. INTRODUCTION AND BASIC TERMINOLOGY

1. INTRODUCTION

This section covers a background motivation for the work, goals of the thesis, an overview of the following sections, and naming and typographical conventions.

1.1. Background Motivation

Extensible Markup Language (XML) [W9] standard was originally projected as a standard format for electronic publishing of documents on the web. However, very soon after its release, XML language has taken place as a general format for interchanging and storing data. However, the great advantages of XML format, such as its openness, human readability, universality, and platform independence, are accompanied with the performance issues when the automated processing of XML documents is required.

Operations over XML data can be executed by common servers with XML aware applications, without any special hardware acceleration for processing XML data. Whereas this approach seems to be sufficient in small organizations, in larger companies, it could cause serious performance bottlenecks in XML processing, especially when the interchanged data should be validated, encrypted, verified, signed, transformed etc. As a result, operations over XML data could slow down the whole chain of executed processes in a company.

The solution could involve adding various load balancers and extra common servers which distributes the processing of XML documents a bit more. However, this approach has its practical limits, due to restriction of resources, still increasing maintenance, data traffic, distribution of security policies, and many other drawbacks.

Other solution includes special appliances that are manufactured to process XML documents and carry out common operations over XML data. Such appliances should accelerate XML processing, at the cost of higher price. Besides the acceleration features, such appliances could be easier to deploy, maintain, with XML-aware security components, support for Web services, and enterprise application integration environments [Erl]. Nevertheless, the same question remains - whether and under which circumstances these appliances are better choice?

1.2. Goals of The Thesis

The main goal of this thesis is to compare the traditional fully software and new hardware accelerated processing of XML documents. For this purpose, the sets of testing scenarios are defined, performed, and compared. Consequently, the results should reveal when the hardware accelerated environment is worth using and, on the other hand, in which situations the common completely software solution is sufficient.

1.3. Conventions

Figure 1 lists formatting styles that give a specific meaning to some parts of the thesis.

Sample text	Meaning
<code>Code</code>	Sample code, configuration file
Important code	Emphasized part of the code

Referenced code	Code part or a figure label referenced from the text
Filename	Filename used in the thesis

Figure 1 Conventions

1.3.1. Hierarchy of Headings

The master thesis consists of the following heading types:

- Part - The whole thesis consists of several parts denoted by a Greek number followed by a dot and a title
 - Sample: I. Introduction and Basic Terminology
- Section - Each part consists of sections denoted by a number followed by a dot and a title
 - Sample: 1. Introduction
- Subsection - Each section can contain subsections depicted as two numbers followed by a title
 - Sample: 1.1. Background
- Chapter - Subsections can further involve chapters symbolized by three numbers followed by a title
 - Sample: 1.3.1 Hierarchy of Headings
- Subchapter - Chapters can have subchapters introduced only by the title of the subchapter
 - Sample: Comparison of XML Schema Languages

These heading types are used when referencing differently nested parts of the thesis.

1.4. Thesis Organization

Part I introduces motivation and goals of this thesis and briefly explains basic XML terminology and technology. Part II describes both testing environments. Subsequently, Part III defines the testing framework and contains definitions of the testing suites. Part IV includes the testing and results comparing. Part V sums up results and delineates possible future work. Finally, Part VI holds additional materials and references.

2. TERMINOLOGY AND TECHNOLOGY OVERVIEW

This section introduces the basic terminology concerning XML language, describes operations over XML documents and usage of XML language in Java.

2.1. XML

Extensible Markup Language (XML) [W9] is a standardized text format derived from Standard Generalized Markup Language (SGML) [W35] and governed by W3 Consortium [W54]. The following chapters describe the basic operations that can be performed over XML data.

Figure 2 depicts a sample XML document. An XML document consists of elements (e.g. an element `firstname`) and attributes (e.g. an attribute `id` of an element `person`). Elements can contain other elements and/or text contents, on the other hand, attributes can involve only text contents. The scope of elements is defined by an opening and closing token with the name of the element closed into angle brackets supplemented with a forward slash in case of the closing token. If the elements are opened and closed properly and for each pair of elements the assertion “If an element A starts before an elements B and the element B starts before the element A ends then the element B is closed before the element A” is valid, the document is well-formed. As a result, the elements of the XML document create a tree structure.

Each XML document contains one root element (e.g. an element `people` in Figure 2) and possibly definitions of other elements inside of the root element. The tree character of an XML document is successfully exploited when processing the XML document. The very first line in Figure 2 declares that it is an XML document of the version 1.0 with UTF-8 text encoding. For a full documentation of XML format, see W3C Recommendation [W56].

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
  <person id="p111">
    <firstname>Jon</firstname>
    <lastname>Bright</lastname>
  </person>
</people>
```

Figure 2 Sample XML Document

2.1.1. Parsing XML Documents

An XML document can be treated as a general text, however, this approach is unnecessarily non-effective. The existence of elements and attributes in XML documents enables the creation of XML-aware parsers. There are two approaches in parsing XML data.

Document Object Model (DOM)

Concept of well-formed XML documents enables recursive processing of an XML document and building a tree structure in a memory, where elements, attributes and other types of XML metadata compose the nodes of the tree. This concept is used by W3C standard DOM Level 3 [W39] and implemented in DOM parsers.

Simple API for XML (SAX)

SAX parsing approach evolves as an opposed variant to a DOM processing in order to reduce the memory footprint. Unlike DOM, the SAX approach has no formal specification, the Java implementation tends to be considered as normative [W40]. Consequently, the SAX approach was implemented in other languages according to implementation in Java.

SAX parser is event-based, therefore, an XML document is stepwise processed and occurrences of defined events (e.g. start of an element, text node occurrence, end of an element) are sent to user-defined callback methods. This processing does not need to build a tree representation of the whole XML document in a memory, hence, the memory consumption is much more lower. However, the SAX parser cannot return to already parsed segments of the XML document, which eliminates the available operations over the XML data parsed in this way (e.g. the children elements of a root element cannot be sorted according to their names).

2.1.2. Validating XML Documents

XML documents contain data that are usually interchanged among different applications. However, without metadata about the content of an XML document, it is hard to prevent or reveal all unexpected, missing, or badly named nodes. What is more, the structure of elements and attributes of an XML document stays only in heads of document designers. Therefore, no other subject (person or machine) can check that a given XML document satisfies some restrictions defined by designers.

In our case, if an XML firewall appliance (see Part II) does not know the right structure of documents that are allowed to go through the firewall, such filtering of XML documents is almost on the same level as in standard firewall where the appliance does not know anything about XML format. Hence, some kind of metadata that describe a structure of an XML document is needed. Then, an incoming XML document can be validated against these rules by the XML firewall and all desiderative machines are aware of the structure of these XML documents. Finally yet importantly, subjects (people or machines) creating XML documents can establish common vocabulary of created XML documents. Hence, the created documents can be denoted as document instances of the XML document describing their structure and thus presenting a template for all other created documents of the same type.

Document Type Definition (DTD)

DTD [W34] was originally included in SGML specification [W35] and later used in XML language specification [W9]. It enables description of a structure of an XML document, however, its expressive power and range of covered data types is limited (see Subchapter “Comparison of XML Schema Languages”).

DTD can be introduced together with an XML document, that it describes, or it can be shipped in a separate file and linked together by a declaration of used DTD files at the beginning of an XML file. DTD contains three base building blocks (key words) designed to describe a structure of an XML document:

- Element
- Attribute
- Entity

Elements are the key building blocks. Each key word element in a DTD describes structure of the element in a respective document instance, which means its name, optional child elements

and their cardinality. Each attribute key word in a DTD involves definition of an attribute used in the respective document instance - its name, a name of element to which it belongs, type of the attribute and a default value and/or whether the attribute is mandatory, optional or has a fixed value. Entities are variables that are used to hold some standard text or special characters (e.g. ‘<’) and they are expanded/substituted with their real value during parsing of the document instance.

DTD supports data types in a limited way. The elements can contain sequence of character data that should be parsed (so-called PCDATA). PCDATA involve entities for expansion and tags that are treated as markup. The attribute data type can be one of the following list (not complete, all can be found on [W34]) :

- CDATA - general character data
- Enumerated list (x | y | ...) - the value must be one of the values in the enumerated list
- ID - unique identification of an element in an XML document
- IDREF - unique identification of another XML element

Figure 3 and Figure 4 show an XML document and a DTD containing records about people. Note that the DOCTYPE tag in the XML document includes the reference to the DTD file `peopleList.dtd` used for validating the XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE peopleList SYSTEM "peopleList.dtd">
<peopleList>
  <person id="p111" currentEmployee="true">
    <name>Jon Bright</name>
    <gender>M</gender>
    <salary>2000</salary>
  </person>
</people_list>
```

Figure 3 XML Document for Explanation of DTD Validation (peopleList.xml)

```
<!ELEMENT peopleList (person*)>
<!ELEMENT person (name, birthdate?, gender?, salary?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT salary (#PCDATA)>

<!ATTLIST person id ID #REQUIRED>
<!ATTLIST person currentEmployee (true|false) "false" >
<!ATTLIST salary note CDATA #IMPLIED>
```

Figure 4 Sample DTD File (peopleList.dtd)

The DTD file defines a root element `peopleList` that contains arbitrary number of the elements `person` (denoted by the following operator ‘*’). Each `person` element consists of a mandatory element `name` and optional elements `birthday`, `gender`, and `salary` (optional elements are denoted by the following operator ‘?’). The next four lines with element definitions depict that the elements `name`, `birthdate`, `gender` and `salary` contain parsed character data (#PCDATA). The element `person` has two attributes. The first one is a mandatory attribute `id` with a value that should be unique among other elements `person` (thanks to data type ID). The second attribute `currentEmployee` indicates whether the person is an employee or not (this attribute can contain only two values - “true”

or “false” - and the value “false” is the default one). The element `salary` can have one optional attribute `note` that contains general character data.

XML Schema

XML Schema [W37] is a W3C Recommendation designed to describe structure of an XML document, which have many advantages over DTD (See Subchapter “Comparison of XML Schema Languages”) and therefore is a preferred way of describing XML documents. XML Schema is far more complex than DTD, therefore, we explain a main idea and comment a chosen example, the rest can be found on [W37].

The expressive power of XML Schema is a superset of DTD. All DTD files can be rewritten to XML Schema with the same meaning. XML Schema Definition (XSD) file is an XML file used to delineate the structure of an XML document (document instance). XSD is capable of defining elements and attributes that can contain simple data types (such as words, numbers, and dates) as well as complex elements that involve attributes and/or other elements arbitrary mixed with text.

XML Schema has plenty of prepared data types that can describe the meaning of the contents of elements and attributes. Such data types can be customized through a set of facets, which allow for example defining and checking the length of a text, a range of admissible numbers, or regular expression that must be matched. What is more, complex data types containing other elements and/or attributes can be created from scratch, can restrict, or extend other types. Whole hierarchies with respect to object-oriented principles of inheritance and polymorphism can be created.

XSD files can be included to other XSD files and reused, therefore, hierarchies of XSD files can be maintained as a common vocabulary of XML documents for the whole company. In order to avoid name collision in XSD files, the concept of namespaces is introduced. Thus, every element in XSD can have special prefix valid for one or more XSD files that logically groups together set of elements. Different sets of elements typically have different prefixes, so that common names of elements (for example “day”) can be used without a danger of a potential name collision. Names of namespaces are typically based on a domain name of a company, which creates them, so that the chance of choosing the same namespace by two companies is minimized.

Last but not least, many integrity restrictions can be defined through the use of elements unique, key, and the XPath [W55] language.

Let us show a sample XML file similar to the XML file in Figure 3 used in Subchapter Document Type Definition (DTD) and its corresponding XSD file.

```
<?xml version="1.0" encoding="UTF-8"?>
<peopleList xmlns="http://www.nappy.cz"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.nappy.cz peopleList.xsd">
  <person id="p111" currentEmployee="true">
    <name>Jon Bright</name>
    <gender>M</gender>
    <salary>2000</salary>
  </person>
</peopleList>
```

Figure 5 Sample XML Document for Explanation of XSD Validation (peopleList.xml)

The differences in comparison with Figure 3 are bolded. The attributes of a root element `peopleList` successively specify the default namespace of this document, the namespace of support elements for XML Schema instances and location of XSD file describing the XML document. Figure 6 shows the XSD file for `peopleList.xml` document and is divided into three segments.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.nappy.cz"
  xmlns="http://www.nappy.cz">

  <!-- Structure of element peopleList - Segment 1-->
  <xs:element name="peopleList">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="person" type="personType" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- Structure of element person - Segment 2 -->
  <xs:complexType name="personType">
    <xs:all>
      <xs:element name="name" type="xs:token"/>
      <xs:element name="birthday" type="xs:date" minOccurs="0"/>
      <xs:element name="gender" type="genderType" minOccurs="0"/>
      <xs:element name="salary" type="salaryType" minOccurs="0"/>
    </xs:all>
    <xs:attribute name="id" type="personIdType" use="required"/>
    <xs:attribute name="currentEmployee" type="xs:boolean" use="optional"
      default="false"/>
  </xs:complexType>

  <!--Decoupled type declarations of elements and attributes - Segment 3-->
  <xs:simpleType name="genderType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="M"/>
      <xs:enumeration value="F"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="salaryType">
    <xs:simpleContent>
      <xs:extension base="xs:positiveInteger">
        <xs:attribute name="note" type="xs:string" use="optional"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:simpleType name="personIdType">
    <xs:restriction base="xs:ID">
      <xs:pattern value="p([0-9]+)/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Figure 6 Sample XSD File (peopleList.xsd)

An element `xs:schema` is a root element of each XSD file and depicts that the schema definition follows (`xs` denotes shortcut of the used namespace and is explicitly mentioned in the following text only if the name without a namespace shortcut is ambiguous). The attributes of the element `schema` successively define the namespace of XML Schema elements (such as `xs:schema`, `xs:element`), `targetNamespace` of XSD file which should correspond to a default namespace of a document instance, and a default namespace of the XSD file. Elements in the default namespace need not be prefixed by a namespace shortcut, its namespace is implicitly taken into account.

Segment 1 of Figure 6 defines an element `peopleList` and its contents. The element `peopleList` contains arbitrary number of elements `person` which is signified by an element `complexType` and its grandchild element `xs:element` with an attribute `maxOccurs` set to unbounded. The attribute `minOccurs` works on the same principle - it determines the minimum number of occurrence of the element. If `minOccurs` and/or `maxOccurs` are omitted, the default value “one” is assumed for both attributes. The child element of the element `complexType` is not important in this case, however, if there would be more element declarations within the element `peopleList`, these elements must occur exactly in the defined order. The declaration of element `person` has decoupled definition to Segment 2 of Figure 6. In fact, each element can be declared and defined on one place, or the declaration can involve attribute `type` referencing the detached definition of the element.

Segment 2 is a decoupled definition of the element `person`. As we can see, the element `person` involves four elements and two attributes. The element `all` indicates that the child elements of the element `person` can appear in an arbitrary order (unlike the element `sequence` in Segment 1). Segment 3 described later contains decoupled type definition of elements `gender` and `salary` and attribute `id`. The rest of elements (`name`, `birthday`) and attributes (`currentEmployee`) have build-in data types, so they are declared and defined on the same place. The mandatory element `name` is a token, which means that it should be a string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces, or multiple spaces. The optional element `birthday` contains a built-in type `date`, therefore only data in a form 1984-07-06 are allowed. The attribute `currentEmployee` should hold boolean values “false” or “true” and the attribute `use` with a value `required` indicates that the attribute is mandatory. Otherwise, if the value of the attribute `use` is `optional` (default value), the attribute is not mandatory.

Segment 3 contains definitions of optional elements `gender` and `salary` and mandatory attribute `id`. The first element `simpleType` holds definition of the element `gender`. The element `gender` is a string data type, but restricted only to two values - “M” and “F” (denoted by an element `restriction` and elements `enumeration`). The element `complexType` describes the element `salary`. The element `salary` contains a positive number from a defined range (see [W37]) and an optional attribute `note` with text contents. The last `simpleType` element in Segment 3 specifies that the attribute `id` from the Segment 2 holds data of type `ID`, which means that the value of this attribute must be unique among other elements containing this attribute. What is more, the value of the attribute `id` is restricted to the values that satisfy the regular expression given in the elements `restriction` and `pattern`.

Comparison of XML Schema Languages

XML Schema language has several advantages over DTD:

- It uses XML syntax and therefore can be parsed, transformed, or validated as other XML documents
- XML Schema has support for namespaces, so that a name collision can be easily prevented
- XML Schema contains plenty of built-in data types, so that the content of elements and attributes can be precisely delineated
- The structure of document can be described more precisely in XML Schema (for example there is no way to specify a mixed content of elements and text in DTD)

On the other hand, the disadvantage of the XML Schema is its talkiness and more complicated syntax.

Other Schema languages

DTD and XML Schema are not the only possibilities for describing structure of XML documents. The other schema languages are (for example):

- Relax NG [W57]
- Schematron [W58]

2.1.3. Transforming XML Documents

In many cases, we would like to transform an XML document A to a document B with a slightly or completely different structure. Extensible Stylesheet Language Transformations (XSLT) is a W3C Recommendation [W59] and language that addresses this needs, hence, a language describing the transformation of the source document to the resulting document.

The so-called XSL stylesheet is governing the transformation process. It contains a set of construction rules, each consisting of a pattern that is tried to be matched against the source document and a template which describes what should be done if the rule is matched. The idea of templates enables us to describe a wide range of XML documents with the same structure by one XSL stylesheet. We explain some of the features of XSLT on an example, rest can be found on [W59].

Figure 7 depicts a source XML document involving a list of employees. Each employee is represented as a single element `employee` with its subelements holding information about the particular employee.

```
<?xml version="1.0" encoding="utf-8"?>
<people>
  <employee>
    <id>0059</id>
    <firstname>John</firstname>
    <lastname>Brown</lastname>
    <city>Prague</city>
    <position>Analyst</position>
  </employee>
  <employee>
    <id>0060</id>
    <firstname>Jack</firstname>
    <lastname>Mill</lastname>
    <city>London</city>
    <position>Tester</position>
  </employee>
</people>
```

```

</employee>
<employee>
  <id>0061</id>
  <firstname>Thomas</firstname>
  <lastname>Lock</lastname>
  <city>Paris</city>
  <position>Analyst</position>
</employee>
</people>

```

Figure 7 Source XML Document for Explanation of XSL Transformations

We create an XSL stylesheet converting the source XML document in Figure 7 to a HTML page with a table of employees working as analysts (specified in the element `position` of the element `employee`). Each row of the resulting table in the HTML page should contain last name of the employee and the city, where the employee works. Figure 8 depicts the necessary XSL stylesheet.

```

[0] <?xml version="1.0" encoding="iso-8859-1"?>
[1] <xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
[2] <xsl:template match="/">
  <html>
    <body>
      <h2>Analysts</h2>
      <table border="1">
        <tr>
          <th>Lastname</th>
          <th>City</th>
        </tr>
[3]     <xsl:for-each select="people/employee[position='Analyst']">
        <tr>
[4]           <td>
              <xsl:value-of select="lastname"/>
            </td>
[5]           <td>
              <xsl:value-of select="city"/>
            </td>
        </tr>
[6]     </xsl:for-each>
      </table>
    </body>
  </html>
[7] </xsl:template>
[8] </xsl:stylesheet>

```

Figure 8 XSL Stylesheet for Creating the Resulting HTML Page

Line 0 declares an XML document and its encoding, because all XSL stylesheets are XML documents as well. Line 1 indicates that the rest of the XML file is an XSL stylesheet of version 1.0. Since the versions 1.0 and 2.0 of XSL stylesheets differ significantly, it is important to introduce the intended one. Line 3 holds a construction rule, with its pattern matching and a template part. This construction rule matches the root element of the source document, indicated by the pattern “/” in the attribute `match`. The attribute `match` can hold arbitrary XPath query which locates part(s) of the source XML document.

The processing of the source XML document begins always from the root element. Firstly, we try to find in the XSL stylesheet a processing rule describing how to behave with the root

element of the source document. If such processing rule does not exist, implicit processing rule is used, which begins to process the direct children of the root element in the source document.

However, we have defined a processing rule matching the root element and therefore its template is executed row by row before doing something else with the rest of the source XML document. Hence, the lines between Line 2 and 3 are copied to the resulting XML, creating a HTML and table header. Consequently, `for-each` element declared in XSL namespace is executed. This construct selects all elements `employees` with a `position` element equal to “Analyst” (employees John and Thomas) and consequently for all these employees, lines between Line 3 and 6 are executed. This lead to creation of two table rows in the resulting HTML page holding for each row contents of the `lastname` element in the first cell and of the element `city` in second cell. The contents of the `lastname` and `city` elements are copied from the source XML document to the resulting document using `value-of` element which executes the XPath query specified within the `select` attribute and returns the acquired value. Finally, the lines between Line 6 and 7 are copied to the output and the resulting table and the whole HTML document are closed. Figure 9 depicts the resulting XML document.

Analysts

Lastname	City
Brown	Prague
Lock	Paris

Figure 9 Resulting XML Document for Explanation of XSL Transformations

2.1.4. Securing XML Documents

The previous standards do not solve any security issues associated with XML data interchange. By default, XML data are sent in a plain form. That is admissible in a small closed network of well-known computers and users, however, if such data are sent over the Internet, securing these transfers is inevitable.

The main goal of securing XML data is preserving integrity and/or confidentiality of (a part of) a message carrying the data. Integrity ensures that a third person does not alter the XML data sent over the wire. Confidentiality hides the XML data so that the content is visible only to the receiver (after decrypting the message). In both cases, asymmetric cryptography is involved. The asymmetric cryptography uses two types of keys, private and public, which form a fixed key pair. Data encrypted with a private key can be successfully decrypted only with the public key from the same pair and vice versa. Therefore, the data encrypted (signed) with the private key of a user A can be decrypted (verified) by a user B only with the public key of the user A. Thus if someone alters the message during the transfer, he does not know the private key of the user A, so the user B (receiver) can easily find out that the message was tampered, because the decoding (verifying) of the message with the public key of the user A will fail. Hence, the integrity of such XML data is preserved (see Figure 10).

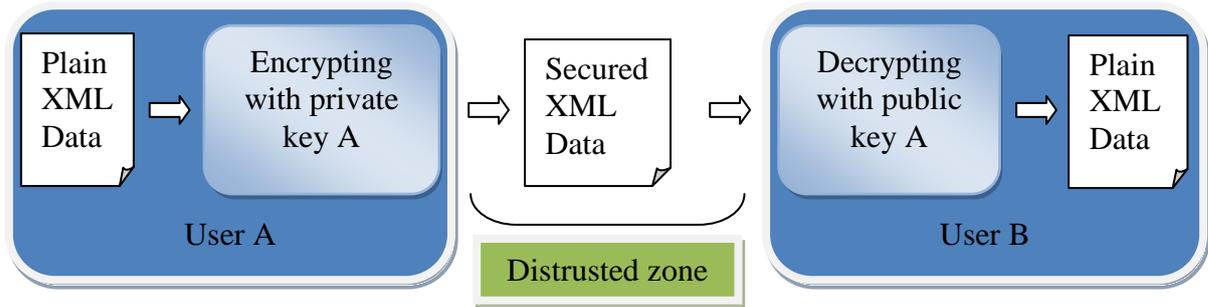


Figure 10 Process of Ensuring Integrity

On the other hand, if the user A encrypts XML data with the public key of the user B, such data can be decrypted only by the user B and his/her private key, which means that confidentiality is preserved (see Figure 11).

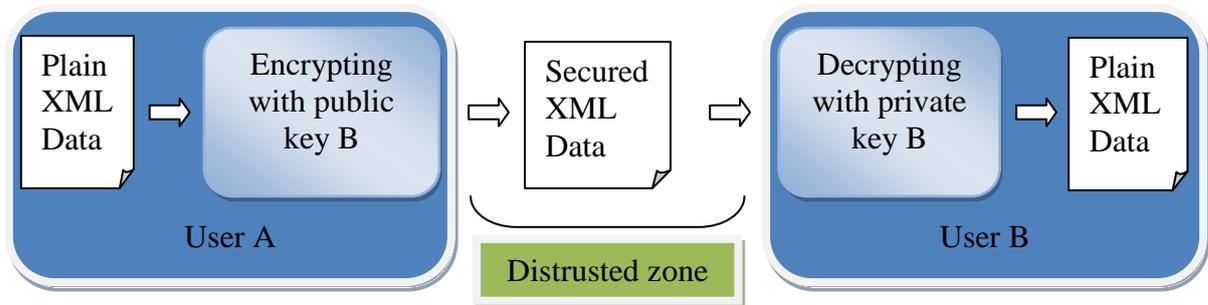


Figure 11 Process of Ensuring Confidentiality

Both techniques can be combined, so that integrity as well as confidentiality of XML data is guaranteed.

Spheres of Activity of a Security Context

A security context is an interval in an XML document transfer that starts with an application of some security principles on the XML document (e.g. the document is encrypted) and ends with the reverse operation (e.g. decryption of the document). The security context can have different spheres of activity (see Figure 12):

- Point-to-point security - Security is enforced only within the adjacent computer nodes
- End-to-end security - Security persists from the initial sender till the desired receiver

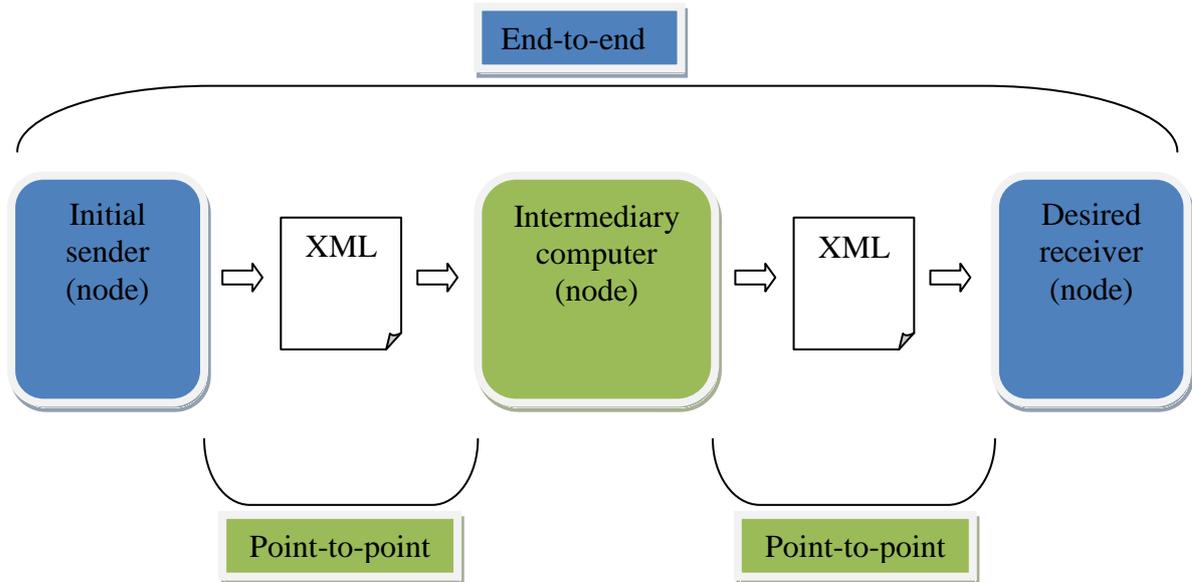


Figure 12 Spheres of Activity of a Security Context

Point-to-point security can be enforced only within two adjacent nodes. If the data should pass one or more intermediary nodes, each intermediary node must create a new security context. If the intermediary node does not need to break the previous security context (for example to access the data encrypted by the previous context), the creation of a new security context unnecessarily increases workload and decreases the overall security, because the intermediary node can be a potential enemy. Typical representative of a point-to-point security is a HTTPS [W60] protocol.

On the other hand, end-to-end security context can be established between the initial sender and the desired receiver, so that no intermediary node needs to break and re-establish the security context. End-to-end security increases throughput as well as security.

All standards defined specially for securing XML work on the end-to-end basis.

Levels of Granularity

Security can be specified on the different levels of granularity:

- Message-level security - Security can be particularized only for an XML document as a whole
- Field-level security - Security can be defined on a part of an XML document (e.g. only desired elements can be encrypted)

In the following text, we describe in more detail the end-to-end field-level security that is native to XML data.

XML Signature

XML Signature is a W3C Recommendation [W16, Ste] for ensuring integrity of XML data. It uses cryptographic hash functions, such as Secure Hash Algorithm version 1 (SHA-1) [W17], and asymmetric algorithms for digital signing and verifying digital signatures, such as Digital Signature Algorithm (DSA) [W18].

XML Signature calculates digest(s) of desired part(s) of XML data (called data object(s)) and digitally signs the acquired digest(s) together with other auxiliary information contained within an element `SignedInfo` using preferred asymmetric cryptographic algorithm. Information about the key that was used to create the XML Signature can be included in the sent XML data. The Figure 13 shows a sample `Signature` element structure:

```
<Signature Id="MySignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="http://example.com/bar.xml#ElementToSign">
      <Transforms>
        <Transform
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
    <KeyInfo>
      <KeyValue>
        <DSAKeyValue>
          <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
        </DSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
```

Figure 13 Example of Using XML Signature

The core element `SignedInfo` contains common as well as data object specific information basis for the signing process. Each `Reference` element within this section describes one digested data object – the element `DigestValue` contains digest of such data object, the element `DigestMethod` specifies method used for digesting such data object and, finally, the element `Transform` depicts chain of transformations that are used to prepare the data object for digesting. The element `Reference` has an attribute `URI` that contains URI [W28] of the referenced data object. The `SignedInfo` element further involves an element `CanonicalizationMethod` that specifies the algorithm used for canonicalization of the `SignedInfo` element before it is actually signed by an algorithm initiated in a `SignatureMethod` element as a part of the whole signature process.

As we can see, the data object(s) that should be signed are actually digested and the digest value is stored in the element `DigestValue`. Consecutively, the whole element `SignedInfo` is signed and the resulting signature value is stored in the element `SignatureValue`.

The element `KeyInfo` contains the key to be used to validate the signature (e.g. a public key). This element is optional, because the receiver can already know such information, hence, it would be redundant information and can be omitted.

For further purposes, it is important to differentiate enveloped and enveloping signature.

In an enveloping signature, the `Signature` element is placed over the signed data object and the data object is identified via the `URI` attribute of the element `Reference` (see Figure 13). On the other hand, an enveloped signature has the element `Signature` as the child element of the signed data object itself. In this case, the element `Transform` is mandatory, so that the element `Signature` can be removed when data object's digest is computed.

XML Encryption

XML Encryption is a W3C Recommendation [W19, Ste] for ensuring confidentiality of XML data objects. Typically, it uses symmetric cryptographic algorithms, such as Triple Data Encryption Standard (DES) [W29] for encrypting XML data objects and, optionally, asymmetric algorithms for encrypting the symmetric key used in XML data encryption.

Figure 14 depicts a sample source XML document based on the example in [W19] that describes one `PaymentInfo` element used for an ATM transaction:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/08</Expiration>
  </CreditCard>
</PaymentInfo>
```

Figure 14 XML Document for Explanation of Encrypting

Figure 15 shows the same XML file but with encrypted content of the element credit card Number:

```
<?xml version='1.0'?>
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>
      <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
        Id='ED'
        Type='http://www.w3.org/2001/04/xmlenc#Content' />
      <EncryptionMethod
        Algorithm='http://www.w3.org/2001/04/xmlenc#tripleDES-cbc' />
      <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
        <ds:RetrievalMethod URI='#EK'
          Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey" />
        <ds:KeyName>John Smith</ds:KeyName>
      </ds:KeyInfo>
      <CipherData>
        <CipherValue>DEADBEEF</CipherValue>
      </CipherData>
    </EncryptedData>
  </Number>
  <Issuer>Example Bank</Issuer>
  <Expiration>04/08</Expiration>
</CreditCard>
</PaymentInfo>
```

Figure 15 XML Document with Encrypted Credit Card Number

The element `CipherValue` contains the encrypted form of the content of element `Number` created by cryptographic algorithm defined in the `EncryptionMethod` element. The `KeyInfo` element contains an element `KeyName` with the name of the key usable for decrypting the encrypted data object. Alternatively, the `KeyInfo` element can contain an element `RetrievalMethod` which specifies URI of an element (see Figure 16) with information about encryption of the key that is actually used to encrypt and decrypt the desired XML data object(s). Element `EncryptedData` encapsulates information about encrypted content and particularize the nature of the encrypted data, e.g. whether the whole element or only its content is encrypted.

```
<EncryptedKey Id='EK' xmlns='http://www.w3.org/2001/04/xmlenc#'>
  <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
    <ds:KeyName>John Smith</ds:KeyName>
  </ds:KeyInfo>
  <CipherData>
    <CipherValue>xyzabc</CipherValue>
  </CipherData>
  <ReferenceList>
    <DataReference URI='#ED' />
  </ReferenceList>
  <CarriedKeyName>Sally Doe</CarriedKeyName>
</EncryptedKey>
```

Figure 16 Information about a Key Used for Encryption and Decryption

The element `EncryptedKey`, referenced by the attribute `URI` of the element `RetrievalMethod` in Figure 15, has a similar structure as the element `EncryptedData`, but holds information about encryption of a key used to encrypt/decrypt XML data object(s). This encryption uses receiver's public key and asymmetric cryptographic algorithms specified in element `EncryptionMethod`, such as RSA [W30], for ensuring that only the desired receiver can decode the element `CipherValue` in the element `EncryptedKey` and, thus, get the access to the key that decrypts the encrypted XML data object in the element `EncryptedData`. The element `EncryptedKey` is typically included in the same XML file as element(s) `EncryptedData`. The element `ReferenceList` contains backward references to `EncryptedData` elements in the XML document. Finally, the element `CarriedKeyName` can hold alternative names to keys.

2.2. XML in Java

XML defines platform independent data format. On the other hand, Java provides cross-platform programming language. This subsection covers the standards for processing XML document in Java.

2.2.1. Java API for XML Processing

Java API for XML Processing (JAXP) is a core Java Community Process (JCP) [W41] recommendation (under a code JSR 206) [W26, McG] for parsing, validating and transforming XML documents. It defines interfaces for processing XML documents and specifies requirements and suggestions for concrete implementations. JAXP (currently in version 1.3) is a successor of Java API for XML Parsing (JSR 005) [W61] which covers only DOM and SAX parsing. JAXP is part of Java 5 API.

Until the formation of JAXP, Java APIs for transformations were proprietary, there were no general conventions. What is more, validation was only a part of a parsing process, the decoupling of validation was not encompassed by JSR 005.

JAXP is designed with a consideration of possible future implementations and offers SAX, DOM, XSLT, XPath and validation pluggability, which means, the custom implementations fulfilling the JAXP interface and conformance requirements can be used. This gives us a chance to try more Java implementations with the same interface.

The following subchapters show simple examples of using JAXP interfaces. In all examples, the exception catching is omitted, so that the examples are more comprehensible.

SAX Parsing

Figure 17 depicts an example of using SAX interface.

```
[1] SAXParserFactory factory = SAXParserFactory.newInstance();
[2] SAXParser parser = factory.newSAXParser();
[2] DefaultHandler handler = new MyApplicationParseHandler();

[4] parser.parse("sourcefile.xml", handler);
```

Figure 17 SAX Parsing Example

Line 1 creates a SAX parser factory. Each SAX parser created from this factory (Line 2) respects the customized settings of the factory. Line 3 defines a handler for catching events which occurs during processing of an XML document. Line 4 executes the parsing task - the specified XML document `sourcefile.xml` is parsed and events about the parsing are sent to the defined handler for further processing.

DOM Parsing

Figure 18 depicts an example of using DOM interface.

```
[1] DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
[2] DocumentBuilder builder = factory.newDocumentBuilder();
[3] Document document = builder.parse("sourcefile.xml");
```

Figure 18 DOM Parsing Example

Again, as in SAX parsing example, Line 1 creates a DOM parser factory. The factory can be customized according to programmer's needs. Consequently, the builder is created from the factory (on Line 2), XML document `sourcefile.xml` is parsed using the builder (on Line 3), and finally, the resulting tree is stored to `document` variable (on Line 3).

XSL Transformation

Figure 19 shows an example on transforming XML documents.

```
[1] TransformerFactory factory = TransformerFactory.newInstance();
[2] Transformer transformer = factory.newTransformer(
    new StreamSource("mystylesheet.xsl")
);
[3] transformer.transform(
    new StreamSource("sourcefile.xml"),
    new StreamResult("resultfile.xml")
);
```

Figure 19 XSL Transformation Example

Line 1 defines a factory for XSL transformation. The factory itself can create XSL transformers (as on Line 2) which are used for transforming the input XML `sourcefile.xml` to the output XML `resultfile.xml` using the stylesheet `mystylesheet.xml`. Again, the factory can be customized so that all transformers created from that factory use the same custom settings.

2.2.2. Java API for XML Digital Signature and XML Encryption

Java Community Process (JCP) [W41] recommendation with the code JSR 105 describes the Java API for XML Digital Signature 1.0 [W20]. The recommendation is in Final Release state since the year 2005 and involves `javax.xml.crypto` package containing common java interfaces for signing and verifying XML documents according to XML Signature W3C Recommendation [W16]. To add, JSR 105 involves a reference implementation of these interfaces. The examples of using `javax.xml.crypto` package are part of the full API documentation [W20]. Java API for XML Digital Signature is part of Java 6 SE.

JCP recommendation JSR 106 describes the Java API for XML Encrypting 1.0 [W21] according to XML Encryption W3C Recommendation [W19]. Unfortunately, JSR 106 recommendation is still in phase of Proposed Public Review Draft, which means, there exists no real implementation of these interfaces. As a result, we have to use another proprietary implementation of XML Encryption in Java.

2.3. J2EE and Servlets

Java Platform, Enterprise Edition [W51, W52, Joh] is a platform for running distributed multitier Java applications using component-based approach [Szy]. Application logic is typically distributed into more components in more tiers of the J2EE architecture. The four typical tiers are Client, Web, Application, and Backend. Moreover, J2EE incorporates transaction support and unified security model which can be employed by the user application.

In the most common web application, client (web browser) in the Client tier sends a request to a Web tier. The components in the Web tier are responsible for displaying the data on the web page, however, the data must be firstly loaded and prepared by components in the Application and Backend tiers. Components of all tiers can be on different machines.

In our case, we use only the Java Servlet component (see Java Servlet Specification [W50]) usable in the Web tier for processing the incoming XML documents (as HTTP payload), applying actions to the documents, and sending responses back to the non-Java client. The advantage of using simple J2EE application is that we do not need to implement threads management, parsing of the incoming and serializing of the outgoing document.

II. TESTING ENVIRONMENTS

3. DESCRIPTION OF TESTING ENVIRONMENTS

This section covers overview of used testing environments. Firstly, we introduce IBM WebSphere DataPower Integration Appliance XI50 for hardware processing of XML data. Further, WebSphere Application Server 6.1 is presented. Finally, auxiliary tools Curl, Apache HTTP Server Benchmark tool, Eclipse, and IBM Rational Application Developer Version 7 are described.

3.1. IBM WebSphere DataPower Integration Appliance XI50

The hardware appliance that is taken into account is IBM WebSphere DataPower Integration Appliance XI50 (hereafter XI50). It is a rack-mountable XML-aware appliance with four network interfaces (see Figure 20 and [W4]).



Figure 20 XI50 – Outer Face of the Appliance

XI50 was originally developed by the DataPower Company, which was acquired by the IBM Company [W1] in May 2005. The appliance is nowadays maintained and further improved by IBM as a part of IBM WebSphere platform [W2].

Except for XI50, IBM produces another two appliances – IBM WebSphere DataPower XML Accelerator XA35 in a green case (hereafter XA35) and IBM WebSphere DataPower XML Security Gateway XS40 covered in a yellow case (in further text XS40). They both support only a subset of operations of their blue “brother” XI50, however, they are cheaper. To be more precise, XA35 appliance is designed as an XSL accelerator without any security and integration features, XS40 has the same features as XA35, adds security features, and still lacks integration features in comparison with XI50 appliance. Our testing scenarios use XI50 as a default appliance, but if XS40 or even XA35 is sufficient for successful accomplishment of a given scenario, this fact is emphasized.

3.1.1. Features of XI50

The main features can be divided into four areas: basic XML acceleration, security, integration, and supporting features.

Basic XML Acceleration Features

Operations over large amount of XML data require appropriate computational capacity because many operations are very time-consuming. Therefore, the hardware acceleration is welcomed. The list of such basic operations involves:

- XSL transformations
- support for XPath queries
- XML Schema validation
- validation of Web services against WSDL

Note that some security features presented in the next section (e.g. XML encryption and XML digital signatures) can also be very time-consuming.

Security Features

Security is sometimes overlooked due to performance issues, but XI50 solves this problem and, what is more, it offers plenty of security features “under the same roof” – from encryption of XML to creation of security tokens for authorizing access to resources behind XI50 appliance. The features include:

- object and message level XML encryption and XML digital signatures
- SOAP [W77] filtering
- XML threats protection (such as XML Denial of Service (XDoS), XML virus protection, dictionary attacks protection etc.)
- authentication of Web services messages using WS-Security [W78] and Security Assertion Markup Language (SAML) [W75], support for WS-SecureConversation [W76]
- authorization for XML messages
- auditing (traceability) of previous authentications, resource allocations
- support for Kerberos [W79], RADIUS [W80], Lightweight Directory Access Protocol (LDAP) [W81]
- ability to process WS-Trust and WS-Federation messages
- Public key infrastructure (support for RSA, DES, Triple DES, AES [W83], SHA, X.509 [W72], CRL [W72], and more)

Integration Features

These features are important when legacy systems are integrated with other systems and/or this appliance is a part of enterprise application integration platform [Er1]. It involves protocols and messages transformations as well as integration with business middleware, enterprise application integration environments, and databases.

- support for wide variety of protocols: HTTP, HTTPS, IBM WebSphere MQ [W74], Java Message Service (JMS) [W73]
- multi-protocol gateway, transformations between these protocols
- Web services proxy
- Universal Description Discovery and Integration (UDDI)
- static and dynamic routing (using XPath, WS-Routing)
- non-XML message (such as binary and flat text) to XML message transformation
- fetching data from external resource (such as HTTP server)
- direct database access - execution of SQL statements against supported databases

Supporting Features

Finally yet importantly, the set of other features, mainly for better overview over development, deployment, and management process, consists of:

- user-defined logging and error catching, user-defined variables for further processing
- service-level management of Web services
- Web Services Distributed Management (WSDM) [W82]

- managing appliance through web GUI, command-line interface, Simple Network Management Protocol (SNMP), SOAP management interface, integrated development environment integration through Eclipse (Chapter 3.3.1) and Altova XML Spy [W6]
- system controlling, detailed probing of processed data
- role-based management (users are working within domains with given authorization)

3.1.2. Typical Use Cases of XI50

Typical use cases of XI50 can be divided into the following scenarios:

- XML firewall – validating and securing XML
- Security gateway and proxy for Web services
- XSL transformations, generating XHTML from XML, generating XML logs
- Transforming non-XML flat file to a data-oriented XML
- Multi-protocol gateway
- Other tasks - generating inserts to databases, fetching content from the given address

XML Firewall – Validating and Securing XML

XI50 acts as a firewall that effectively processes incoming and outgoing traffic on content basis (it knows XML). Hence, XML data getting through XI50 appliance can be filtered or validated against a schema, digitally signed files can be verified, encrypted files decrypted, and company security policies (even using external appliances such as a RADIUS server) can be specified and enforced. As a result, incoming and outgoing XML files are accepted or rejected according to inner company needs. XI50 also contains build-in protection against known XML threats (see Chapter 3.1.1).

Security Gateway and Proxy for Web Services

Here, XI50 plays role of a security gateway for Web services. Similar operations as in the previous scenario are performed, however, applied to requests and responses of Web services. Family of WS-Security standards is used to authenticate SOAP requests according to a wide range of authentication tokens, to enforce integrity and/or confidentiality of SOAP request (or even part of the request), and to apply security policies similarly to the previous scenario. In addition, XI50 can act as a proxy for Web services, which yields in hiding Web service's backend.

XSL Transformations, Generating XHTML from XML, Generating XML Log

XI50 can be used for fast processing of a large amount of XML data that have to be transformed before they are presented or moved on. Such transformations are very often significantly time-consuming and, consequently, they cause performance issues.

Transforming a Non-XML Flat File to a Data-oriented XML Document

XI50 supports transforming non-XML files to XML files and vice versa. For describing the structure of non-XML files, proprietary Flat Files Description (FFD) files specified and maintained by Contivo [W14] are used. Contivo is involved in semantic integration of data within business network. It produces software for defining common vocabulary repositories, managing conversions between XML files and automatically generating XSL transformations according to vocabulary repositories and user-defined mapping between source and target XML files.

A FFD file describes the structure of a given flat file and it is used to convert a flat file to an XML file. Figure 21 and Figure 22 depict a source flat file and an appropriate FFD file:

```
Multilayer Technology GMBH CO. KG  20020677  USD  US  A
Smart Modular                      20000689  USD  US  A
```

Figure 21 Example of a Flat File

```
<? xml version="1.0" encoding="UTF-8"?>
<File name="Prism">
  <Group name="record" delim="\n" maxOccurs="unbounded">
    <Field name="VendorName" length="35"/>
    <Field name="VendorNumber" length="10"/>
    <Field name="Currency" length="5"/>
    <Field name="Country" length="3"/>
    <Field name="SyncIndicator" length="1"/>
  </Group>
  <Literal value="\n"/>
</File>
```

Figure 22 Flat File Description (FFD) File for the Above Flat File

According to this Flat File Description file, records in the flat file are delimited by the new line and number of such records is unbounded. Each record is represented as an element `record` in an output XML and has five child elements with the names specified by the attribute name of elements `Field` in FFD. An Attribute `length` of the element `Field` in FFD describes the length of a text in the flat file that is pasted into appropriate element's value in the output XML file.

The FFD file can have far more complex structure. Unfortunately, it is not an open format, so details are not publicly available.

Multi-protocol Gateway

XI50 can accept different protocols and convert them to desired protocol(s). To be accurate, HTTP, HTTPS, IBM WebSphere MQ, Java Message Service (JMS) are supported. Therefore, XI50 can, for example, listen on HTTP and HTTPS protocol's port and convert such request(s) to MQ messages that are subsequently sent to a business network.

Other Tasks - Generating Inserts to Databases, Fetching Content from a Given Address

XI50 can generate inserts to a wide range of supported databases. It is not a main feature of XI50 but it simplifies processing of data. Furthermore, XI50 can fetch content from a specified address, e.g. from an HTTP server.

3.1.3. Connection to a Business Network

XI50 can be connected to different segments of a business network. It can work at the edge of business network (thus in so-called demilitarized zone) just behind the standard firewall, where it acts as an XML firewall, a security and multi-protocol gateway, and/or a Web services proxy. Demilitarized zones can be also between two divisions within the same organization, but the purpose would be similar to the previous case. Furthermore, XI50 can work on special operations within the internal business network, such as generating XHTML pages, re-routing Web services, transforming XML files between two data sources etc. (For more scenarios, see Chapter 3.1.2).

Nevertheless, what is common for all of these scenarios, XI50 can be connected in two basic modes – in a proxy and a coprocessor mode. In real scenarios, especially when serving as gateways and entry points to business networks, where availability of such appliances is crucial, XI50s are typically used in a pair with a load balancer distributing traffic between them. For clearness in the following pictures, only one XI50 is shown.

Proxy Mode

In this case, XI50 acts as a proxy (see Figure 23), i.e. all communication (even if not needed) goes through XI50. What is more, network infrastructure need not be changed (we just split a cable and connect XI50 to both new ends).

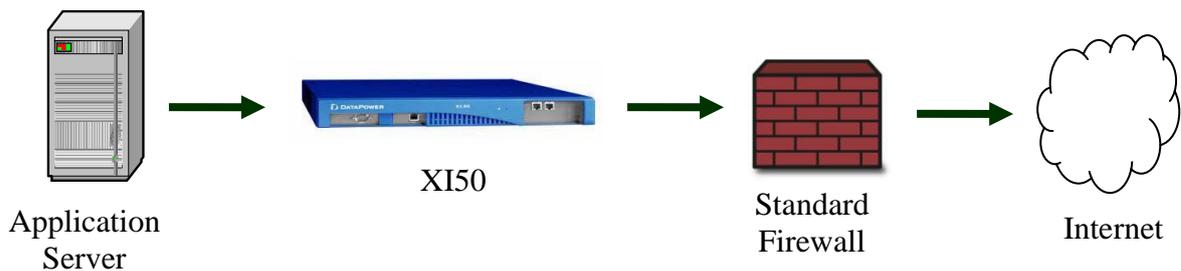


Figure 23 Proxy Mode of XI50 Connection to a Network

Coprocessor Mode

Conversely, coprocessor mode of XI50 connection yields more traffic on an application server linked to XI50, because of sending data to and receiving processed data from XI50 (see Figure 24). However, XI50 does not need to probe all packets as shown in proxy mode. The proxy mode is preferred, because the coprocessor mode yields more traffic overhead due to one extra network hop.

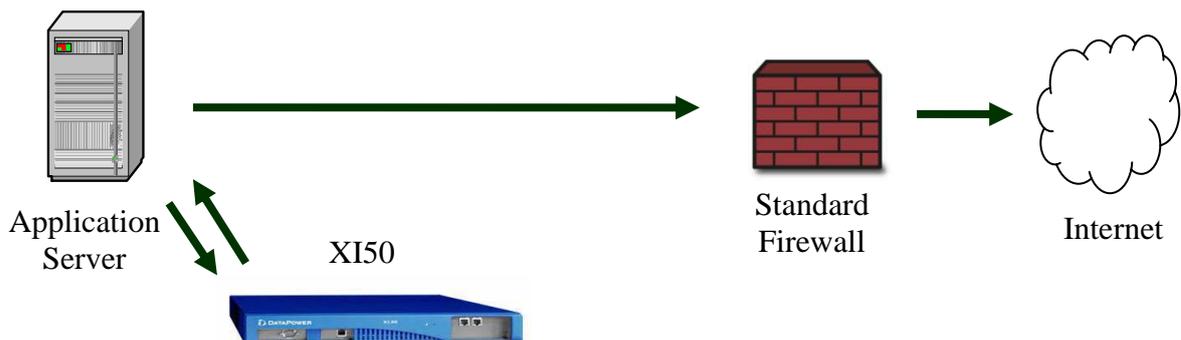


Figure 24 Coprocessor Mode of XI50 Connection to a Network

3.1.4. Web-based GUI

XI50 can be comfortably managed by a web-based interface. Each user logs in under the specific domain, which grants him rights according to business needs.

The main window offers a portal to main services, such as Web Service Proxy and XML Firewall. Further, it allows access to logs and monitoring tools, as well as to the inner file system and export/import possibilities (see Figure 25).



Figure 25 Web-based GUI - Main Window

The most interesting part of the web-based GUI comprises named object store accessible under the left menu field “Objects” (see Figure 25). It consists of a wide range of objects - building blocks - used to create all application services. Objects definitions are decoupled as much as possible and the objects can be referenced from potentially many other objects, which eliminates redefinitions of (parts of) objects. The objects needed for our testing are described more precisely in a definition of the testing framework in Subsection 4.5.

Complete Web-based GUI reference can be found on [W4] (free registration required). Apart from web-based GUI, Command Line Interface (CLI) or SOAP-based XML Management Interface is accessible.

3.1.5. Internal Device Structure

There is not much available information about the real internal structure of XI50. However, according to the documentation, web-based GUI system information, and experience, it should have a main processing engine for processing and dispatching requests, an accelerator for securing XML, XML-aware memory addressable by XPath queries, and an add-on accelerator card for further accelerating and decoupling XML parsing and schema validation from the main XML processing engine). XI50 has no movable secondary memories, but almost 4GB of flash memory. All IBM appliances are firmware updatable. An interconnection of inner components is unfortunately not publicly available.

3.1.6. Approximate Price

According to the hardware announcement from March 28, 2006 [W15], the lowest model XA35 costs about \$35,000, the middle model XS40 about \$65,000, and, finally, the highest model XI50 \$75,000. The prices above are for appropriate appliances in their base versions with a third-generation (XG3) processing technology. Appliances with the new fourth-generation (XG4) processing technology (contain a special add-on accelerator card for decoupling XML parsing and schema validation from the main XML processing engine) and/or with some extra features such as IBM Tivoli Access Manager and/or maintenance support cost more.

3.1.7. Other Appliances

IBM WebSphere DataPower appliances are not the only existing appliances for accelerating XML processing, however, there exists no directly comparable product. The market with similar appliances, involves:

- Cisco ACE XML Gateway (produced by Cisco [W10]) - XML firewall, approximately comparable with XS40 appliance with some integration features.
- SecureSpan XML Networking Gateway (Layer 7 Technologies [W12]) - XML firewall with PCI-e SSL and XML accelerator cards, approximately comparable to XS40 with integration support.

Generally, appliances specializing in XML processing are not so broadened in Czech Republic as in Western Europe or America and the true value of these appliances is nowadays not fully appreciated. Nevertheless, the situation is likely to meliorate in the following years, because standard firewalls tend to be insufficient and, what is more, overlooked security principals can have deplorable consequences.

Firstly, we are working with IBM WebSphere DataPower appliance (XI50) because, as written above, it is nowadays not easy to have an access to such sort of hardware appliances. Secondly, DataPower appliances were one of the first appliances in this category. In addition, its tight relation with IBM WebSphere application server gives us opportunities to compare software and hardware processing of XML data by products of one vendor.

3.2. IBM WebSphere Application Server v 6.1

IBM WebSphere Application Server v 6.1 [W2] (in the following text Application server or WAS) provides a common environment and programming model for running applications that are insulated from the underlying hardware, operating system, and network. WAS is built on J2EE standard, offers tools for building, deploying, installing, running, securing, load balancing, and maintaining J2EE applications. WAS can also serve as a web server, endpoint of Web services, security authority, and/or integrator of legacy applications with new ones. WAS is the key part of IBM WebSphere software platform that helps to build service-oriented architecture [Erl] and integrates applications over a common message broker. WAS 6.1 implements J2EE Specification v1.4 [W53] and Java Servlet Specification 2.4 [W50].

3.2.1. Architectural Overview

Figure 26 shows simplified architecture of an Application Server that is important for our thesis, for the full architectural overview and all supported standards, see [W2, W38].

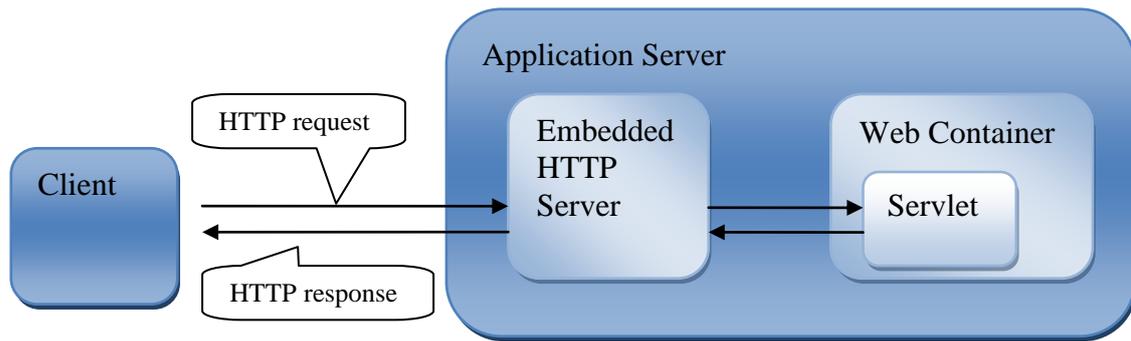


Figure 26 Simple Architectural Overview of Application Server

The HTTP request comes from a client to an embedded HTTP server on an Application Server and is sent to an appropriate Servlet according to local path of the request URL and adjusted servlet mapping. Web Container is a container for J2EE web components, such as Servlets (see 2.3). The Servlet processes the request and creates response that is sent back to the client.

3.2.2. Installation and Profiling

Files associated with WAS can be divided into two areas:

- Product files - set of WAS binaries shared by instances of Application Servers
- Configuration files - set of customizable data files called profiles, which govern the individual instances of Application Servers

Hence, profiles enable creation of different instances of WAS with different configurations. Obviously, only one instance is running at one time during our testing so that the memory and other resources are not wasted.

3.2.3. Administration of the Application Server

WAS installation involves a web browser-based tool for managing the profiles of Application Server (see Figure 27).

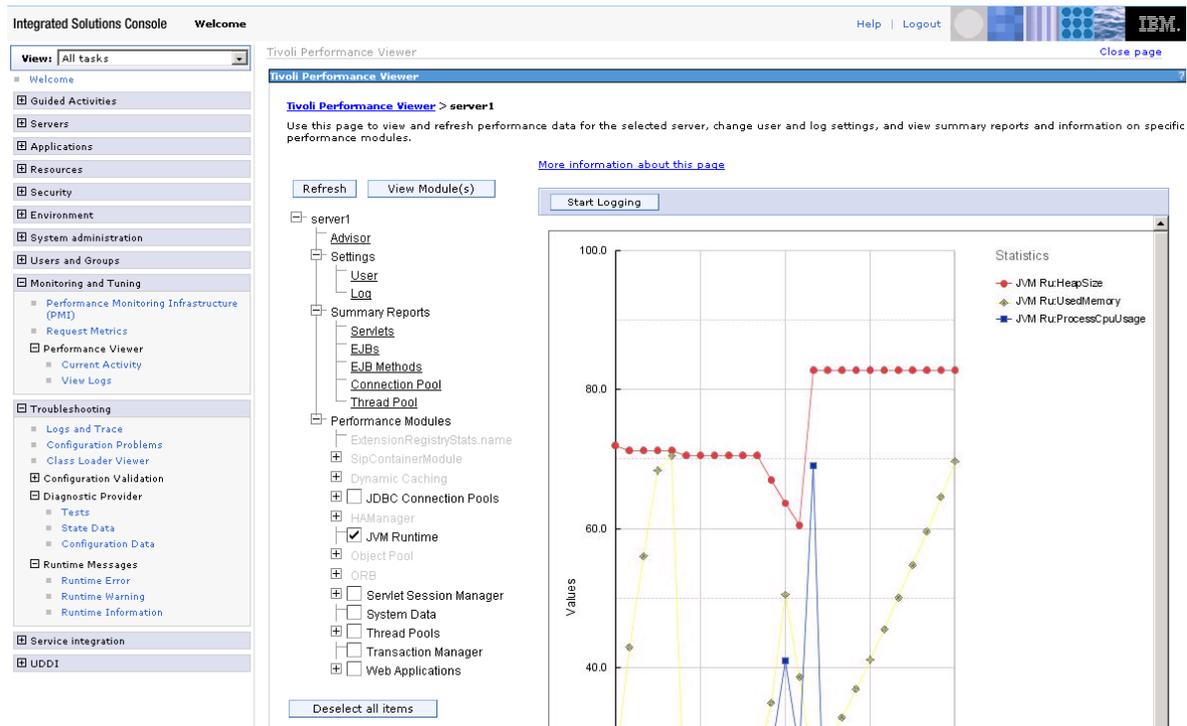


Figure 27 Administrative Console for IBM WebSphere Application Server

The important functionalities of the administration tool are:

- managing configuration of the Application server (e.g. used JVM parameters)
- observing the status of installed applications, starting, stopping, and installing applications
- monitoring the installed and started applications and tuning the performance of them
- logging and tracing possibilities

3.2.4. Approximate Price

The price of IBM WebSphere Application Server version 6.1 depends on the target server, where the WAS is installed. For our four cores Xeon, the license costs 8250\$ (2062.5\$ per one core). IBM WAS can be downloaded and evaluated free of charge for two months.

3.2.5. Other Application Servers or SW Solutions

There are many J2EE Application Servers on the market, for example:

- BEA WebLogic Server [W64]
- JBoss Enterprise Application Platform [W65]
- Sun GlassFish Enterprise Server [W66]

All J2EE compatible application servers can be successfully used instead of IBM WAS. Nevertheless, when utilizing IBM WAS and XI50 appliance, we have a chance to compare two products from the same vendor.

3.3. Auxiliary Tools

In this subsection, auxiliary tools Eclipse, IBM Rational Application Developer Version 7, Curl, Apache HTTP Server Benchmark tool, and Windows Server 2003 Performance tool are described.

3.3.1. Eclipse Version 3

Eclipse [W8] is an open source Java integrated development environment and platform for further tools from different vendors, because the functionality of Eclipse is easily extended through plug-ins of third-party vendors.

Eclipse is the core of IBM Rational Application Developer products. Hence, we do not use Eclipse directly, but through IBM Rational Application Developer 7 (See Chapter 3.3.2)

3.3.2. IBM Rational Application Developer Version 7

IBM Rational Application Developer Version 7 (RAD) is built on Eclipse and is specialized for developing applications for IBM WebSphere Application Server with many helpful additional features.

RAD can start, stop, and manage IBM WebSphere Application Server 6.1, the key part of our software testing environment. Applications developed in RAD and deployed on target Application server can be debugged directly from RAD IDE.

RAD can sniff traffic between client and Application Server and display, resend, or modify content of requests and/or responses sent to the server.

IBM Rational Application Developer 7 is freely available for two months on [W1] (after free registration).

3.3.3. Curl

Curl [W7] is a command line tool for transferring files to a remote server, supporting wide variety of protocols, such as FTP, HTTP, HTTPS, and TELNET. Thanks to curl, it is possible to send files over supported protocols to desired destination and get responses to desired output. We use curl to launch the testing scenarios for the first time, so that we can verify the received data (printed on the defined output). As the data are sent in a “--data-binary” mode, formatting is preserved and the response is readable. For further testing, Apache HTTP server benchmarking tool (AB) is used (see Chapter 3.3.4). To be compared with Curl, AB does not slow the testing process with writing response to the output, enables concurrent requests, and collects statistical data.

An example of using curl to send XML data (stored in the file `queen.xml`) to a given destination is shown in Figure 28 . The response is sent to a standard output:

```
curl --data-binary @queens.xml http://localhost:9082/XML/EntryServlet
```

Figure 28 Curl Sample Request

One known problem of this stuff is a very bad support for sending large files for processing. Still, XI50 can download needful large files from a web server, rather than wait for files to be uploaded.

3.3.4. Apache HTTP Server Benchmark Tool (AB)

AB [W33] is a command line tool for benchmarking a HTTP server. In comparison with Curl, no other protocols apart from HTTP are supported, however, this tool provides useful information for measuring throughput of a given batch of HTTP requests. Figure 29 shows a typical usage of an AB command from Windows Command Line tool:

```
ab.exe -n 10 -c 1 -p queens.xml http://localhost:9082/XML/EntryServlet
```

Figure 29 AB Sample Request

The meaning of fragments of the example in Figure 29 is as follows:

- “-n 10” - The number of requests dispatched to the defined destination
- “-c 1” - The number of concurrent requests
- “-p queens.xml” - The file sent in one request
- “http://localhost:9082/XML/EntryServlet” - The destination of sent files

The final report contains useful information for measuring the tests, especially the total size of transferred data, time per request and requests per second measures, and time taken before the testing was completed. An example of the AB report is depicted in Figure 30.

```
Server Software:      WebSphere
Server Hostname:     192.168.201.1
Server Port:         9082

Document Path:       /XML/EntryServlet
Document Length:     16045 bytes

Concurrency Level:   1
Time taken for tests: 0.406250 seconds
Complete requests:   10
Failed requests:     0
Write errors:        0
Total transferred:   161910 bytes
Total POSTed:        162130
HTML transferred:   160450 bytes
Requests per second: 24.62 [#/sec] (mean)
Time per request:    40.625 [ms] (mean)
Time per request:    40.625 [ms] (mean, across all concurrent requests)
Transfer rate:       388.92 [Kbytes/sec] received
                    389.74 kb/s sent
                    778.94 kb/s total

Connection Times (ms)
                min          mean          [+/-sd]        median        max
Connect:        0           6           7.7            0            15
Processing:     15          34          20.5           31            78
Waiting:        15          32          21.5           31            78
Total:          15          40          18.3           31            78

Percentage of the requests served within a certain time (ms)
 50%    31
 66%    46
 75%    46
 80%    62
 90%    78
 95%    78
 98%    78
 99%    78
100%    78 (longest request)
```

Figure 30 AB Sample Response

Again, as with the Curl command line tool, instead of pushing large HTTP files to a desired destination, it is better to download the desired files from a HTTP server.

3.3.5. Windows Server 2003 Performance Tool

Windows Server Performance tool is used for measuring important resources (e.g. processor, memory, hard disk utilization) during the testing in the SW environment.

III. TESTING - DEFINING MODELS AND METRICS

4. OVERVIEW OF THE TESTING SUITES

This section covers an overview of the testing suites, the interconnection of the testing components and the definition of our testing framework.

4.1. Naming Conventions

For further purposes, if we talk about processing actions or actions, we mean applying an XML technology (such as XSLT) on given XML data. Each action has its input and output contexts which specify inputs and outputs to or from the action. A processing rule denotes a precisely defined progression of actions. A processing policy contains one or more processing rules. If we talk about data, or testing data, it means the input XML data to a processing policy. Measures denote the measured values during the testing (e.g. requests per second measure).

A test case run represents an application of a processing policy on the given testing data and writing down associated measures. A test case symbolizes repeated test case runs with the same processing policy and testing data. A testing scenario consists of one or more test case(s). A testing group denotes one or more testing scenarios that logically go together. Finally, testing suite presents collection of testing groups.

The testing hierarchy signifies a testing suite which has one or more testing groups which involve one or more testing scenarios which contain one or more test cases and underlying test case runs. A model represents a definition of a processing policy and testing data incorporated in a testing hierarchy.

A testing framework is a set of tools and programs that enables the testing in fully software (SW) environments and hardware accelerated (HW) environments.

4.2. Process of Searching Suitable Collections of Testing Scenarios

This subsection covers a background for choosing our testing suites.

4.2.1. Way to the “Onion” Testing Suite

In a SW environment, the complexity of processing XML data tends to cause performance bottlenecks in a business network. On the other hand, in a HW environment, XI50 appliance should be a device with a wire-speed XML processing, thus the XML documents transferred over the wire should use the capacity of the network, as if there is no such XML processing appliance.

Therefore, the one thing that should be investigated is the break point, after which the performance of SW environments begins to lose breath rapidly and becomes useless. The second question is, if an assertion about wire-speed XML processing in the HW environment is true under every condition.

To answer these questions, incremental stress testing, starting with a very simple testing scenario with one action, and gradually adding other actions, can show us these break points and overall usefulness of both environments. What is more, it can lead us to interesting partial results (between adjacent testing scenarios and both environments) as well as global results

that can show when and how quickly HW and SW environments lose power with more complicated testing scenarios.

Let us call this kind of a test as the “Onion” testing suite, because the adding of actions to processing rules of incessantly tougher and tougher testing scenarios symbolizes the layers of a growing onion.

Sample Processing Rule of the “Onion” testing suite

Figure 31 depicts a sample processing rule of a testing scenario of a testing group in the “Onion” testing suite.

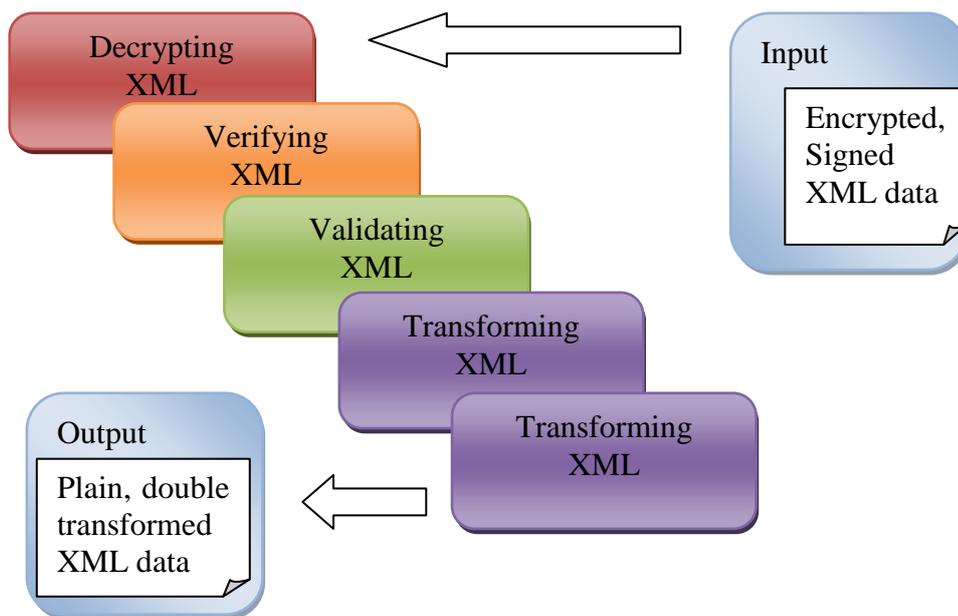


Figure 31 Sample Processing Rule of a Testing Scenario in the “Onion” Testing Suite

It is obvious, that an order of the actions in a processing rule is ambiguous, however, on the other hand, not all permutations of actions shown in Figure 31 give meaningful results. For example, validating fully encrypted XML data does not make sense, because in an encrypted XML document, original nodes are not accessible.

Although some processing rules have meaningful orders of actions, they are not practically useful. For example, XML validation can precede signature verification, but the validation can be worthless if the verification of the XML document failed.

More details about used actions are in Subsections 4.5 and 4.6.

4.2.2. Way to the “Flat” Testing Suite

The “Onion” testing suite is a great stress test, however, there are other more basic questions to be answered, such as what testing data should be chosen for the “Onion” testing suite? What engines should be exploited? How much the slightly different sizes of testing data of one testing scenario influence the gathered measures?

Such questions lead us to the second, so-called “Flat” testing suite, which decouples testing of individual actions over XML data into separate groups of testing scenarios.

Firstly, the “Flat” testing suite should be a non-stress test, so that the behaviour of a wide range of different sizes of testing data and actions with slightly different settings can be compared and analyzed without bothering with a concurrency when processing XML data.

Secondly, in the HW environment, we have at our disposal only XI50, but in the SW environment there exist multiple engines for processing the desired actions over XML data (for example a couple of different XSLT processors). Therefore, the “Flat” testing suite gives us a chance to compare the performance of different engine implementations, select the best one, and compare it with XI50 in the “Onion” testing suite.

To conclude, the “Flat” testing suite deeply analyzes individual actions over XML data and, consequently, purveys results for better selection of processing rules and testing data of the “Onion” testing suite. In other words, thanks to the “Flat” testing suite, the “Onion” testing suite can have significantly less amount of testing data and therefore it can really aim to concurrency and stress testing with a solid knowledge of behaviour of the testing data utilized in the “Flat” testing suite.

Conformance testing for determining compatibility with XML standards is not part of our testing scope.

4.3. Outline of the Testing Hierarchy

The following tree outlines the higher levels of the testing hierarchy - the testing suites and its testing groups with a short description. Complete testing hierarchy explanation can be found in Sections 5 and 6:

- The “Flat” testing suite (Section 5) covers the most common techniques in XML processing, except for XQuery [W67], which is not supported on XI50 and thus its performance cannot be measured and compared. XML technologies are tested individually.
 - Parsing testing group - There are two approaches in processing XML documents. The first one involves buffering a whole XML document and executing actions on that buffered object in an operational memory. This approach is referred as DOM parsing technique. The second one successively buffers small parts of an XML document, executes the desired action on the part of XML document and discards the buffer. Let us refer this approach as Streaming parsing technique. Both techniques are described more deeply in Subsections 4.5 and 4.6. This testing scenario covers parsing of different sizes of testing data using both techniques (DOM and Stream).
 - Validating testing group covers validating of XML documents against XSD and DTD files, using different sizes of input testing data and schema files with a different level of complexity.
 - Transforming testing group contains various XSL stylesheets applied on various input data. It indirectly includes processing of XPath queries.
 - Securing testing group includes encrypting and digital signing of a part of/whole XML document using different cryptographic algorithms and subsequent decrypting and verifying of encrypted and signed documents.
- The “Onion” testing suite (Section 6) combines different actions to one processing rule.

- Auction testing group is a complex testing group from a stock market environment putting together validations, transformations, encryptions and digital signatures. It creates HTML reports about buyers on the auction.
- CSVOutput testing group is a complex testing group generating secured comma separated exports from an XML database of people. It contains validations, transformations, decrypting and signature verification, where all validating and transforming actions of the processing rule can be processed in a streamed way.

4.4. Architecture of the Testing Framework

The architecture of components participating in our testing framework differs in the HW and SW environment, however, it is common for both testing suites.

In the SW environment (Figure 32), a client sends a HTTP request with an input XML document through AB tool to the network interface of a Server. The request is redirected to an IBM WebSphere Application Server 6.1 entrance port and successively sent to a J2EE application entrance point and to an appropriate Servlet according to URI of the request. Subsequently, the logic of the Servlet invokes proper processing rule, executes actions on input data and creates HTTP response with output data from the processing rule. HTTP response is sent back to the original client.

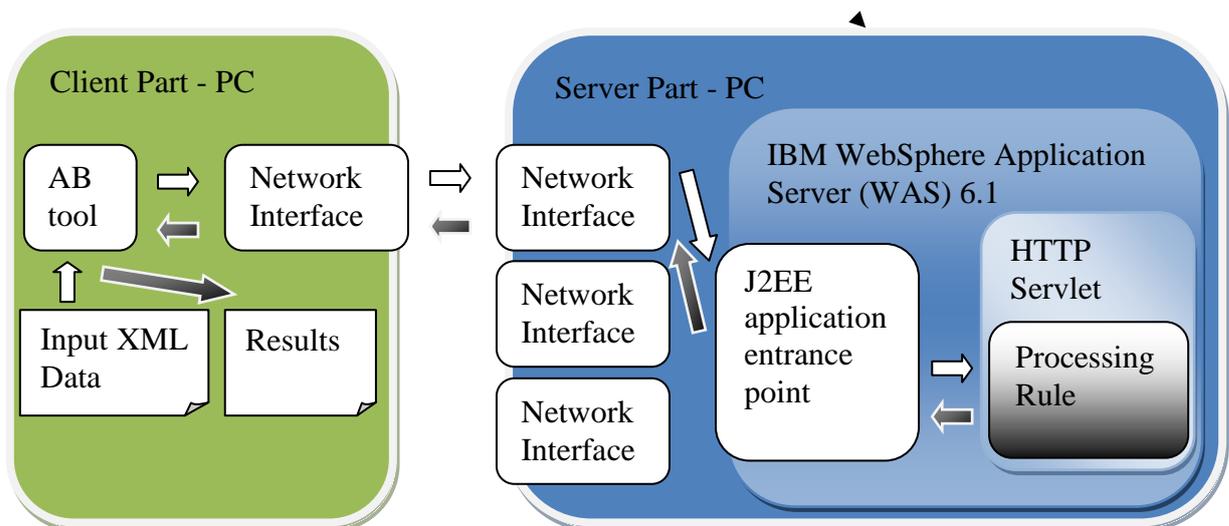


Figure 32 Architecture of the Testing Framework in the SW Environment

On the other hand, in the HW environment (Figure 33), the input XML document is not posted directly to an XI50 appliance, however, the AB tool sends only a simple HTTP GET request and the desired input XML document is downloaded from an Apache HTTP Server.

To be more precise, when the HTTP GET request reaches network interface of XI50 appliance, it is redirected to a defined XML Firewall (according to a port in the HTTP request). Later on, the defined processing policy of the XML Firewall is entered and the processing rule defined for the client-to-server traffic is invoked (denoted as “Processing rule (C-S)” in Figure 33). This processing rule is very simple, it just forwards the request to a backend (HTTP server) defined in the XML Firewall settings. When the request attains the Apache HTTP server, the desired XML document is fetched and downloaded to XI50 appliance.

As the response reaches XI50 appliance, the same XML Firewall and its processing policy is called, nevertheless, the processing rule for traffic in a server-to-client direction is utilized (denoted as “Processing rule (S-C)” in Figure 33). The server-to-client processing rule is in fact the executive processing rule similar to a processing rule in the SW environment in Figure 32. The server-to-client processing rule is applied to the downloaded input XML document and an output XML document is created and sent back to the original client.

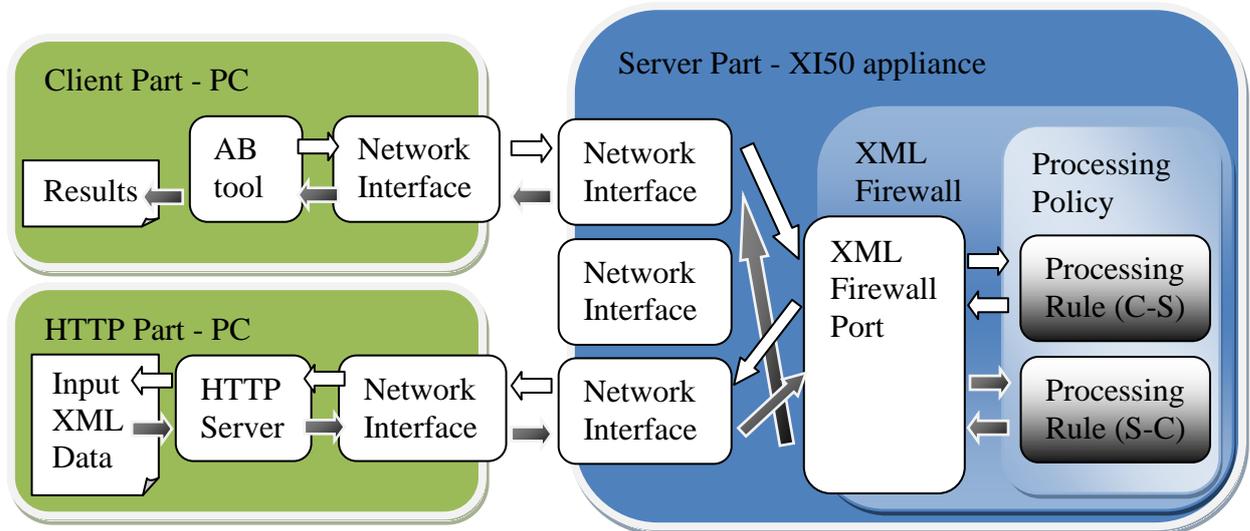


Figure 33 Architecture of the Testing Framework in the HW Environment

The main difference between SW and HW approaches is an extra step (downloading files from the HTTP Server) in processing requests in the HW environment. Nevertheless, this corresponds to a real-world scenario, where the utilization of XI50 appliance connected to a business network in a proxy mode adds one more component to a network topology, and, thus, brings extra network overhead caused by an extra network hop. Therefore, it makes sense to compare the SW environment with two components (client, server) and the HW environment with three network components (client, server, XI50 appliance). The resulting measures are directly comparable and show the gain or loss in processing XML documents with or without connected XI50 appliance.

Subsections 4.5 and 4.6 describe the server part of our testing framework in the HW and SW environment (i.e. “blue part” in Figure 32 and Figure 33). The main goal is to clarify how the processing rules with its processing actions are defined in both environments.

4.5. Server Part of the Testing Framework in HW Environment

This chapter explains the XI50 settings essential for building server part of our testing framework shown in Figure 33.

An XML Firewall is a base object used for processing incoming requests and outgoing responses. As we can see on a sample XML Firewall object summary screen in Figure 34, each XML Firewall object has its name (Firewall Name), type (Firewall Type), associated XML Manager, policy for processing requests and responses (Firewall Policy), and definitions of Back End (e.g. Application Server) and Front End (A network interface on XI50) targets. The possible settings and settings used in our testing framework are described in the following chapters.

Configure XML Firewall

General | Advanced | Stylesheet Params | Headers | Monitors | XML Threat Protection

Apply | Cancel | Delete | Clone | Export | View Log | View Status | Show Probe | Validate Conformance | Help

XML Firewall Service status: [up]

General Configuration

Firewall Name
HTTPtest *

Summary
MT XML Firewall

Firewall Type
Static Backend *

XML Manager
MT_XSLT_XMLManager + ... *

Firewall Policy
HTTP_RowsOnion_DEC_VER_XSLT_VAL_XSLT + ... *

URL Rewrite Policy
(none) + ...



Back End

Server Address
9.138.237.80 *

Server Port
80 *

SSL Client Crypto Profile
(none) + ...

Response Type
XML

Response Attachments
Strip

Front End

Device Address
0.0.0.0 Select Alias *

Device Port
50900 *

SSL Server Crypto Profile
(none) + ...

Request Type
Pass-Thru

Figure 34 Configuring XML Firewall

4.5.1. Front End and Back End Sections

The `Front End` section of the XML Firewall definition in Figure 34 contains a `Device Address` definition of a network interface which accepts the requests (0.0.0.0 denotes arbitrary network interface on XI50). The `Device Port` specifies a port on which the XML Firewall is listening, therefore all requests directed to this port are processed by this XML Firewall object. The `Request Type` within the `Front End` section specifies a type of incoming requests and can contain the following values:

- `XML` - XML Firewall is expecting a message with general XML data, the content of the incoming request is parsed and prepared for further processing by a processing rule.
- `SOAP` - A request message is a SOAP message, the XML Firewall parses the request and automatically validates the SOAP message
- `Non-XML` - The message does not contain XML data, nevertheless, it is buffered by XI50 appliance for further usage.

- `Pass-Thru` - The incoming message is passed through the XML Firewall “as-is”, message content is not buffered.

The `Pass-Thru` is the best choice for a request type in our testing framework, because the only purpose of the request is to download desired XML document for further processing by a server-to-client processing rule.

The `Back End` section has similar fields as `Front End` section and defines an address and a port of the back end system (such as an Application Server or HTTP server) to which the filtered request is sent for further processing. In our case, the `server address` contains IP address of a machine with Apache HTTP Server.

4.5.2. Firewall Type

The `Firewall Type` involves the following possibilities:

- `Loopback-proxy` - Defines an XML Firewall object which receives requests from a client and sends a response back immediately. No back end is called (i.e. the `Back End` section of the XML Firewall definition in Figure 34 is disabled).
- `Static-backend` - The `Back End` section of XML Firewall is enabled and specifies a fixed definition of a back end system (such as HTTP server or Application server) which processes an incoming request and creates a response.
- `Dynamic-backend` - Similar to `static-backend`, however, the target back end system is specified dynamically according to the incoming request.

In our testing framework, we used `Static-backend` choice, which enables redirecting a request to Apache HTTP Server.

4.5.3. XML Manager

The `XML Manager` can be shared among many XML Firewall objects and contains settings influencing the way in which the data are processed. For example, it specifies a security restriction on the size, depth, number of elements of a processed XML document, caching policies of used XSL stylesheets, settings of XG4 plug-in acceleration card, and whether the streaming is enabled or not. For particular settings of the XML Manager object in our testing framework, see Chapter 7.1.1.

4.5.4. Firewall Policy

A `Firewall policy` holds a set of processing rules which define the actions that are applied to the requests coming from a client (and possibly further dispatched to a server) and to the responses coming from a server back to a particular client. Figure 35 depicts a sample processing policy screen.

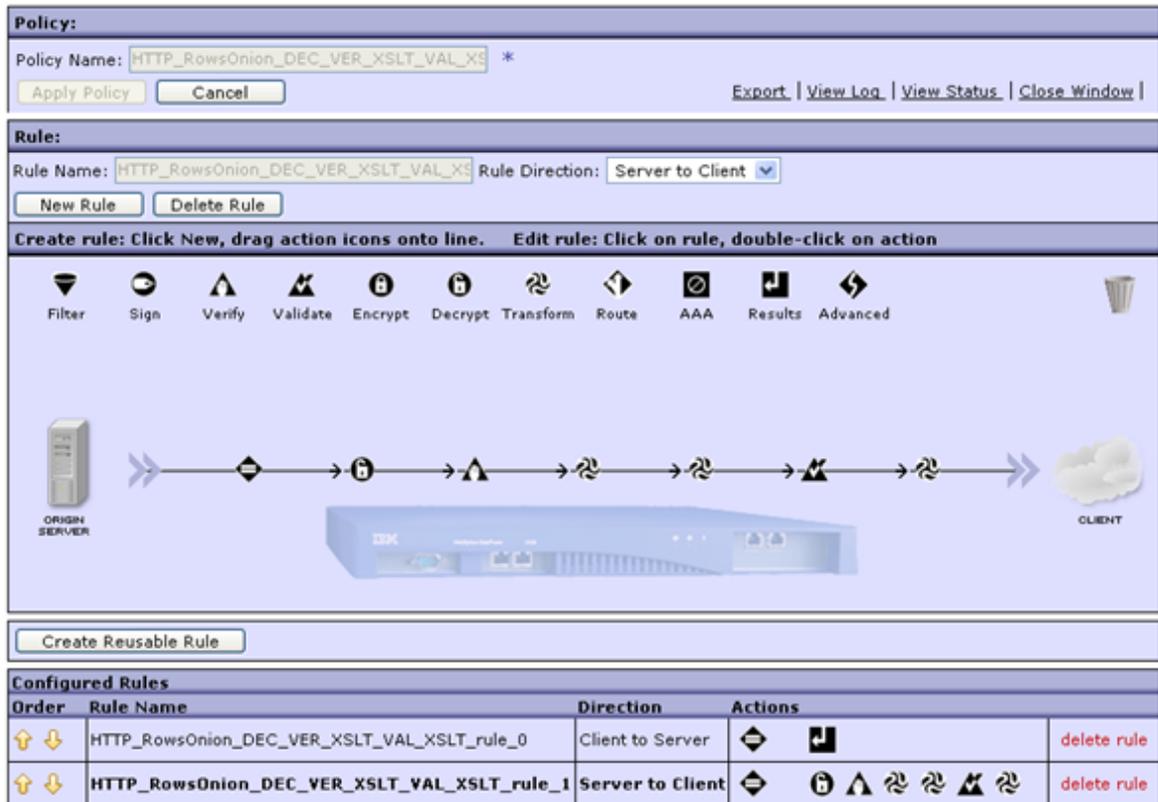


Figure 35 Sample Processing Policy with a Set of Processing Rules

The bottom part of a policy screen contains a set of processing rules governed by this processing policy. There are three types of processing rules according to the impacted data Direction:

- Client to Server (C-S) - A processing rule is applicable to the data coming to the XI50 appliance for the first time (typically from a client)
- Server to Client (S-C) - A rule is relevant for the data coming to the XI50 appliance for the second time (typically as a response from a server)
- Both Directions - A rule can be applied to all incoming data.

If there are more processing rules, the first rule that can be used on the incoming data is applied. That is a rule that describes the appropriate direction and whose Matching action is satisfied (see Chapter 4.5.5).

Each processing rule governs a sequential set of processing actions, which are applied to the incoming data. A queue of actions is for all rules depicted in the bottom part of a firewall policy screen next to the name and the direction of the processing rule.

A middle part of the firewall policy screen in Figure 35 gives a detailed view of the queue of actions of the selected processing rule and enables easy adding, removing, and editing actions of the rule (the list of available actions is depicted in Subchapter “Supported XML Actions”). The selected rule in Figure 35 has seven actions (Matching, Decrypting, Verifying, Transforming, Transforming, Validating, and Transforming) which are consequently applied to a response coming from a server (because it is a S-C processing rule).

Finally, the top part of the firewall processing policy screen depicts a unique name of the processing policy.

Supported XML Actions

The following list contains names and icons of actions that are important for our testing framework (sorted alphabetically). The details of actions are described in Sections 5 and 6.

- Decrypting  - Action for decrypting XML data according to XML Encryption W3C Recommendation [W19]
- Encrypting  - Action for encrypting XML data according to XML Encryption 1.0 W3C Recommendation [W19]
- Signing  - Action for signing XML data according to XML Signature 1.0 W3C Recommendation [W16]
- Transforming  - Action for executing XSL transformations according to XSLT 1.0 W3C Recommendation [W59]
- Validating  - Action for validating data against XML Schema 1.0 W3C Recommendation [W37]
- Verifying  - Action for verifying signed XML data according to XML Signature W3C Recommendation [W16]

The following two auxiliary actions are used:

- Matching  - see Chapter 4.5.5
- Resulting  - see Chapter 4.5.5

Validating and Matching actions are always streamable and a subset of Transforming actions is streamable as well (see Chapter 7.1.2).

There are far more actions, such as SYNC action for synchronizing several branches of parallel execution, FETCH for fetching a document from a target system. All actions are documented on [W4].

Supported Input and Output Contexts to the Actions

The supported input contexts are as follows:

- INPUT
- (USER_DEFINED)
- PIPED
- NULL

The supported output contexts involve:

- OUTPUT
- (USER_DEFINED)
- PIPED
- NULL

Each processing action (except of Matching action) has its input and output contexts. INPUT context specifies reading of an XML document directly from the incoming request/response. On the other hand, OUTPUT context specifies a serialization of an XML document to the outgoing response. The other contexts are used to control the data flow between actions in a processing rule.

NULL context indicates that an action does not require/create input/output at all. Furthermore, the output context can have arbitrary name (e.g. “OUTPUT_OF_XSLT”, “MY_CONTEXT”), which can be used as an input context to the following actions of a processing rule belonging to the same processing policy. What is more, such context name can be specified as an output context in a C-S processing rule and as an input context to an action in a S-C processing rule. This type of context is denoted as (USER_DEFINED). If a PIPED output context is specified, the output of an action is directly passed to the next action in the same processing rule with a mandatory PIPED context as an input context.

The main difference between arbitrary named context and PIPED context is in the memory requirements. Every named context stores its actual state of the processed data (even if the data sizes are tens of megabytes) in a memory together with tens of variables describing the origin of the data, used firewall policy, processing rule etc. Moreover, the named context created in a C-S processing rule is valid in an S-C processing rule as well. Therefore, if C-S and S-C processing rules of one processing policy contain both several processing actions, the memory requirements are roughly equal to the sum of sizes of all metadata in all contexts, which can be several times higher than the size of the input data itself. The stored context metadata are garbage collected only after the last processing action of the processing policy is executed. On the other hand, PIPED context metadata are marked as a garbage immediately after the data are accepted by the next action in a processing rule with a PIPED input context.

PIPED context cannot be used in all actions, because it behaves like a local streaming inside of XI50 appliance, therefore, only Transforming and Validating actions can use PIPED input and output contexts. What is more, PIPED context must be used if the whole processing rule contains more than one action (except of Matching action) and should be streamable. If not specified otherwise, PIPED contexts are used in our testing framework where possible.

4.5.5. XML Firewall Used in Our Testing Framework

Except of the Firewall Policy settings and little changes in the XML Manager settings, all XML Firewall settings shown in Figure 34 depict the real values used in all our testing scenarios in the HW environment.

The changes to XML Manager are specific for each testing scenario and are described together with the particular testing scenario in Sections 5 and 6. The processing policy is also testing scenario specific, however, all processing policies contain two processing rules. Whereas the S-C processing rule (the second one in Figure 35) is testing scenario specific (see Sections 5 and 6), the C-S processing rule is the same for all testing scenarios in our testing framework and is depicted in Figure 36.



Figure 36 Common Client to Server Processing Rule in HW Environment

The only goal of the C-S processing rule is to forward the request to the HTTP Server (see Figure 33) so that the desired XML document can be downloaded and processed by the scenario specific S-C processing rule. The C-S processing rule involves two actions:

- **Matching action** - This action is by default in all processing rules on the XI50 appliance and specifies a document set, which is accepted by the processing rule. The match condition can be based on URLs, HTTP headers of incoming XML documents, XPath expressions, or error codes, however we do not need to filter the incoming documents in any way. Therefore, in all our testing scenarios, the matching pattern is set to ‘*’, which means that all incoming data are accepted by this processing rule. The matching action has no input or output contexts, it just clarifies which documents are accepted by the processing rule. The detail of Matching action is as simple as Figure 37 shows.



Figure 37 Detail of Matching Action

- **Results action** - Results action buffers the processed XML document and sends it as a whole to the special OUTPUT context when ready. If the processing rule does not define any other processing actions (except for Matching action), the Results action must be used. In our common C-S processing rule (see Figure 36), the Results action has a special INPUT context as an input context and OUTPUT context as an output context. The middle part of Figure 38 is not used, we simply copy the data from input context INPUT to the output context OUTPUT. The definitions of input and output contexts are important - these definitions are in all actions except for the Matching action.

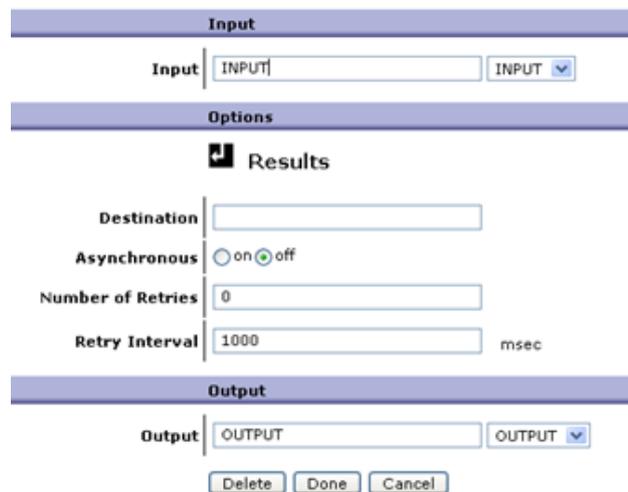


Figure 38 Detail of Results Action

In Section 5, all S-C processing rules for particular testing scenarios are specified, together with details of each processing action. In Section 6, symbolic notation as in Figure 31 is used for depicting processing rules, with emphasized input and output contexts of the processing actions.

4.6. Server Part of the Testing Framework in the SW Environment

This chapter defines in a more precise way the server part of the testing framework in the SW environment depicted in Figure 32.

The server part of the testing framework consists of a servlet-based J2EE application developed in Rational Application Developer (see Chapter 3.3.2) and is deployed on IBM WebSphere Application Server (see Subsection 3.2). The application processes incoming requests with XML documents as HTTP payload and creates responses. Hence, it makes the same work as XML Firewall object defined in Subsection 4.5. The processing of an XML document inside the Servlet is managed by a Queue Manager and consists of a processing rule (or a queue of actions) that is applied to the input XML document. The Queue Manager must support individual execution of actions as well as pipelining of actions.

4.6.1. Processing Rules

Figure 39 shows a sequence diagram of execution of three actions. The first action (symbolized as Action 1) is carried out individually, thus the action is initialized and executed before going to the next action in the queue of actions (processing rule). The second and the third action (denoted as Actions 2 and 3) are pipelined, hence, both these actions are firstly initialized and consequently Action 2 is executed. This yields to execution of Action 3 which is directly supplied with outputs of Action 2. Results of Action 3 are the overall results of these two pipelined actions. Therefore, N pipelined actions can be comprehended as one complex compound action.

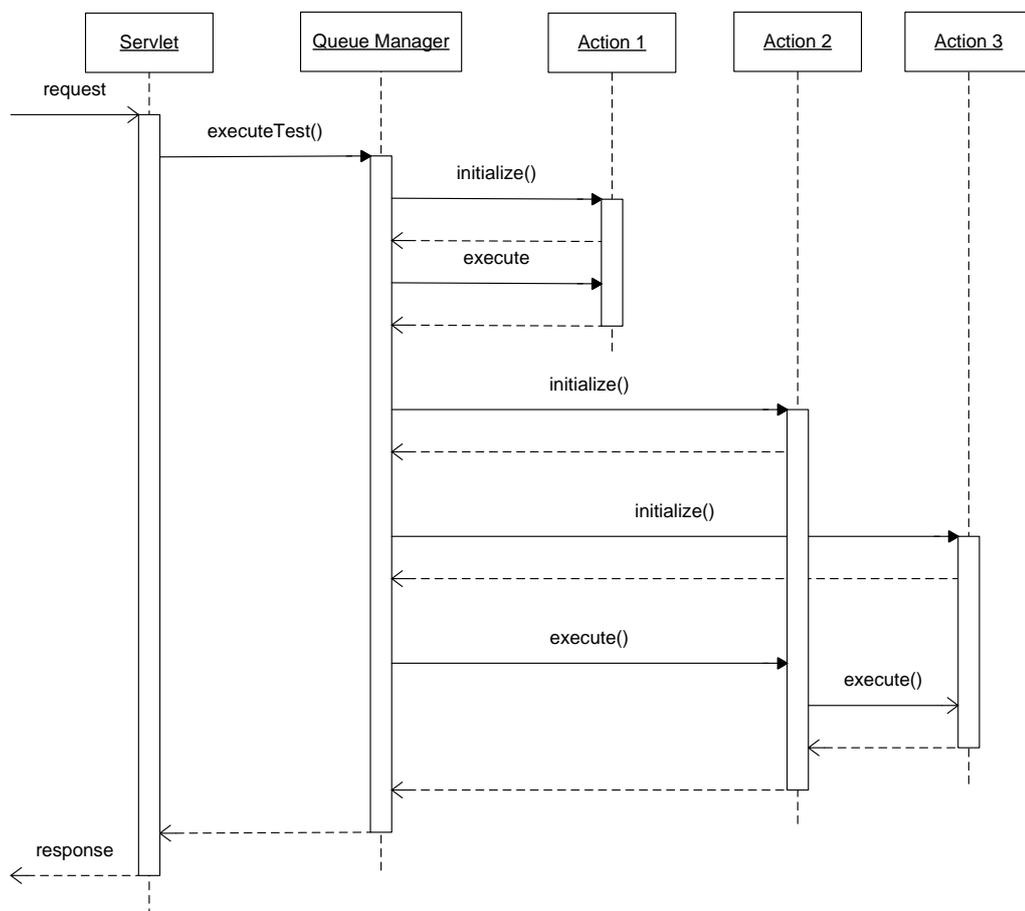


Figure 39 Sequence Diagram of Processing Individual and Pipelined Actions

Processing of actions is the core functionality of our testing framework. In the following subchapters, the supported XML actions, possible inputs and outputs to the actions, and feasible samples of processing rules are introduced.

Supported XML Actions

The following XML actions are used in our testing framework in the SW environment (sorted alphabetically):

- Decrypting - Action for decrypting XML data according to XML Encryption W3C Recommendation [W19]
- Encrypting - Action for encrypting XML data according to XML Encryption 1.0 W3C Recommendation [W19]
- Stream parsing (this action cannot be decoupled on XI50) - Action for parsing XML data in a streamed way according to SAX API 2.0.2 [W67]
- Signing - Action for signing XML data according to XML Signature 1.0 W3C Recommendation [W16]
- Transforming - Action for executing XSL transformations according to XSLT 1.0 W3C Recommendation [W59]
- Validating (XML Schema and DTD validation) - Action for validating data against XML Schema 1.0 W3C Recommendation [W37] and DTD [W34]
- Verifying - Action for verifying signed XML data according to XML Signature W3C Recommendation [W16]

In addition, TreeWrapper action is used for building and serializing DOM tree according to DOM Level 3 Specification [W39].

Stream parsing and Validating action can be streamed, all other actions need to access the whole document. Transformations can be streamed if a streamable engine is utilized.

Each processing action has its unique name, the definitions of inputs and outputs to the action and other attributes, such as a name of XSL stylesheet used for the particular XSLT transformation.

Supported Input and Output Contexts to the Actions

The supported input contexts to the actions are as follows:

- INPUT
- DOM
- PIPED

The supported output contexts involve:

- OUTPUT
- DOM
- PIPED
- NULL

An action that has INPUT as its input context directly reads an XML document from a HTTP request. On the other hand, if the action has OUTPUT as its output context, the output of the action is directly serialized to HTTP response payload. The processing rule must contain exactly one action with its input context defined as INPUT and exactly one action with an output context set to OUTPUT.

DOM input/output of an action denotes that the action is processed as an individual action, hence, the DOM tree is input/output to the action. If an action has defined DOM as an output context, the following action must have a DOM context as an input context which takes as an input the DOM tree created by the previous action. A PIPED output context of an action can be used in conjunction with the following action in the processing rule that must have set its input context to a PIPED context, in which case, the output of one action is directly sent to the input of the following action without the need of building intermediary DOM tree. PIPED approach is based on emitting and processing SAX events.

NULL output context is applicable only to the action that generates no result, such as Verifying of a signed XML document.

In all cases, an output context of an action must correspond to an input context of the following action, except for INPUT, OUTPUT (both must appear only once in a processing rule) and NULL. If the NULL context is used as an output context of an action, the last not-null output context of the previous actions in a processing rule is used as an input to the following action.

If the whole processing rule should be streamable, it must contain only streamable actions and if the processing rule consists of more than two action, these actions must use only INPUT, OUTPUT and PIPED contexts.

To add, we do not support (USER_DEFINED) contexts as in the HW environment, because an output of one action is directly use as an input to the following action.

Individual and Pipelined Actions

A queue of individual actions uses a DOM tree in an operation memory to store intermediate results between adjacent actions. This approach is perfectly fine for actions which must operate on the whole XML document, such as various security operations (Encrypting, Signing action) or XSL transformations working with the whole XML document (e.g. stylesheet involving a global sorting).

However, there are actions that do not inevitably need the whole XML document in a memory to be successfully executed (such as all Validating, Parsing and some Transforming actions). Pipelining of actions enables to redirect an output of one action directly to the input of the following action, so that no intermediate structure is built in a memory. The pipelining utilizes the fact that JAXP interfaces of Transforming and Validating actions support SAX events as inputs and can emit SAX events as an output.

Processing Rule Definitions

Figure 40, Figure 41, and Figure 42 illustrate sample processing rules with different input and output contexts of actions in the processing rule (denoted by a rectangle with an arrow). The colour rectangles depict one processing action with a type of action (e.g. Transforming), support file necessary for executing the action (e.g. an XSL file), and, optionally, a particular java engine implementing the action in a brackets after the action type.



Figure 40 A Processing Rule with a Transforming Action

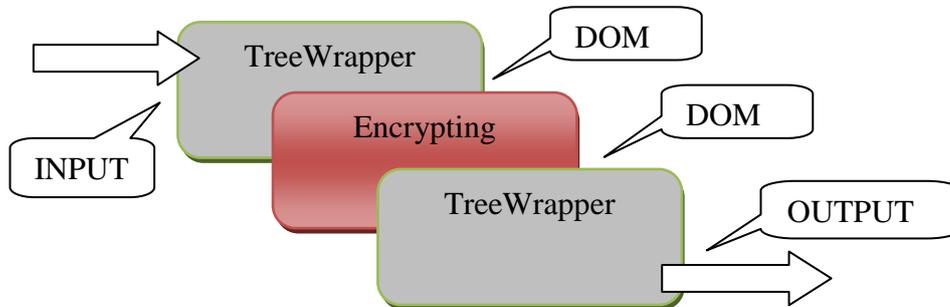


Figure 41 A Processing Rule with an Encrypting Action and TreeWrapper Actions

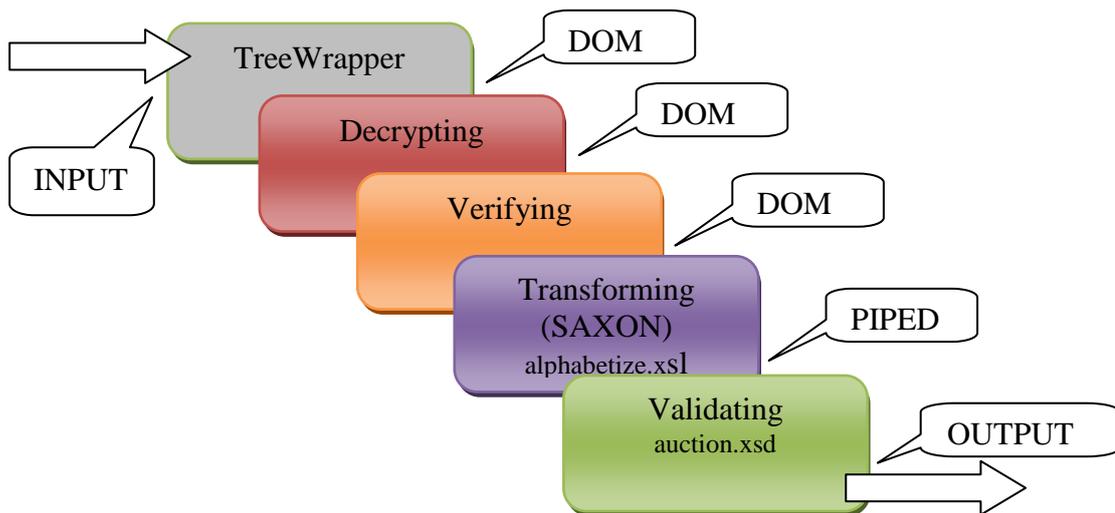


Figure 42 A Complex Rule with DOM and PIPED Input and Output Contexts

Figure 43 depicts a Java code that defines the same processing rule as Figure 40.

```
//Name of a processing rule
[1] String name = "Transforming_XMark_SAXON_alphabetize";

//Creating processing rule
[2] List<ActionMetaData> aMetaList = new ArrayList<ActionMetaData>();

{
    //Creating a processing action
    [3] ActionMetaData aMetaData = new ActionMetaData();
    [4] aMetaData.setActionName(name + "action");
    [5] aMetaData.setActionType(TypeOfIncomingAction.XSLT_SAXON);
    [6] aMetaData.setActionFile("alphabetize.xsl");
    [7] aMetaData.setActionInput(TypeOfInput.INPUT);
    [8] aMetaData.setActionOutput(TypeOfOutput.OUTPUT);
}
```

```

    //Adding the action to the processing rule
    [9] aMetaList.add(aMetaData);
}

//Adding the processing rule to a map of all processing rule
[10] testCaseRepository.put(name, aMetaList);

```

Figure 43 Definition of a Processing Rule

Line 1 holds a unique name for the whole processing rule and policy as well (in the SW environment, processing policy always involves only one processing rule). Line 2 declares a processing rule as a list of processing actions and allocates memory on the heap. Consequently, Line 3 declares and allocates memory for a new processing action.

Line 4 specifies a unique name for a processing action (based on the name of the processing rule). Line 5 introduces a type of action based on supported XML actions and used engines (see Section 5). Line 6 holds a file which is necessary for the given type of action (such as XSL stylesheet, XSD file). Lines 7 and 8 successively define input and output contexts of the action and finally, Line 9 adds the processing action to the processing rule.

Line 10 inserts the whole processing rule to a map of all processing rules, so that it can be quickly accessed when necessary. Lines 3-9 can be repeated more than once to specify more actions in a processing rule. For example, in a case of processing rule in Figure 42, Lines 4-9 are utilized five times (once for each processing action) in a java code definition.

Particular settings of all processing rules in the SW environment in Sections 5 and 6 are depicted using colour rectangles instead of Java code. A colour rectangle (e.g. in Figure 40) holds exactly the same information as bolded lines (Lines 5-8) in Figure 43, however, in a more transparent way.

4.7. Measuring the Tests

Important question is, how should be the measures collected.

The first possibility is sniffing of the beginnings and endings of individual actions or whole processing rules, but this has several disadvantages. On XI50, the debug mode can be turned on, so that specific details about actions can be marked down, or an extra dummy XSL transformation writing down details about processed actions can be added as an extra action to a measured processing rule. Unfortunately, both these solutions are too time consuming and the time spent by these solutions is too variable, so that the overhead for such measuring is hardly estimatable. What is more, it could reduce the speed of processing the whole processing rule several times. Moreover, both these solution do not consider the network traffic overhead. In the SW environment, the same problems with sniffing individual actions can occur.

The other possible solution involves measuring the processed actions together with a network traffic overhead. This gives us a chance to utilize the different architectures of the SW and HW environments. What is more, it allows us to measure the overall time spent on more than one request and enables measuring of stress tests. Moreover, if sufficient number of requests is sent for processing, the measures gathered for the whole batch of requests neutralizes the deviations in processing times of single requests (e.g. influence of CPU interruptions, network delays, memory fetches).

To sum up, we use the batch measuring of our test cases using the AB measures gathered in an AB report.

4.7.1. Measures

The following key measures are observed:

- Throughput (KB/s, MB/s)
- Requests per Second (R/s)
- Time per request (ms)

The throughput is the major key measure, because the throughput is tight to the input and output sizes of XML documents as well as to the processing time. Therefore, it is a good way to characterize a real-world performance.

The following additional measures can be obtained:

- Time taken for tests (s)
- Length of request/response (Bytes)

In addition to the specified measures purveyed by the AB tool report, the following complementary measures are tracked:

- CPU utilization (%) - Usage of a general processing unit (CPU)
- Workload (Only on XI50) (%) - Workload of the whole device, including general processing unit and all accelerators.
- Maximum memory usage (%)

5. DEFINING THE “FLAT” TESTING SUITE

This section covers a definition of the testing scenarios of the “Flat” testing suite. The goal of the “Flat” testing suite is to examine individual actions over XML data in a fully software as well as hardware accelerated environment.

5.1. The Outline of the Testing Groups and Testing Scenarios

This testing suite consists of the following testing groups and testing scenarios:

- Parsing XML data (for more details see Subsection 5.2)
 - Par_DOM
 - Par_STREAM
- Validating XML data (see Subsection 5.3)
 - Val_DTD_BASE
 - Val_XSD_BASE
 - Val_XSD_REDUCED
 - Val_XSD_EXTENDED
- Transforming XML data (see Subsection 5.4)
 - Tra_XSLTMark
 - Tra_XSLTMark_L
 - Tra_XSLTMark_XL
 - Tra_XSLTMark_XXL
- Securing XML data (see Subsection 5.5)
 - Sec_ENC_ENC_RSA2_3DES
 - Sec_ENC_ENC_RSA2_AES256
 - Sec_ENC_ENC_RSA2_AES128
 - Sec_ENC_ENC_RSA_3DES
 - Sec_ENC_ENC_RSA_AES256
 - Sec_ENC_ENC_PART_RSA2_AES256
 - Sec_ENC_DEC_RSA2_3DES
 - Sec_ENC_DEC_RSA2_AES256
 - Sec_SIG_SIG_DSA_SHA1
 - Sec_SIG_SIG_RSA_SHA1
 - Sec_SIG_SIG_RSA2_SHA1
 - Sec_SIG_VER_DSA_SHA1

5.2. Testing Group: Parsing XML Data

The goal of this testing group is to compare DOM and Stream approaches in processing XML documents. What is more, it is a reference testing group, because all other testing groups use implicitly DOM or Stream XML processing. This testing group should also answer questions of memory consumption, effectivity of garbage collection policies and changes in measured metrics related to the increased memory capabilities.

5.2.1. Testing Data and Testing Hierarchy

As to the testing data, we choose XMark [W48] and its generator of synthetic XML documents. XMark generator is scalable, sizes of generated XML documents fluctuate from tens of kilobytes to tens of gigabytes. The structure of these XML documents is realistic, filled with fictitious data from a stock market environment. The generated XML documents tend to be data-oriented with elements containing longer text (e.g. notes to the transferred

stock market transactions, constructed from a combination of thousands of most frequently used words from Shakespeare's plays).

Unlike generators creating XML documents with a random structure, XMark has an advantage of generating XML documents which are valid against an invariable XSD file. Moreover, XMark documents have reasonably complicated structure, so that they can be used in other testing scenarios, which demand some level of complexity, such as in XML Schema validations.

Figure 44 shows the list of used XMark files with the custom names and sizes.

Filename	Size (KBs, MBs, GBs)	
Test0.0	26.5 KB	Very Small
Test0.0001	29.5	
Test0.00015	33.6	
Test0.0002	38.1	
Test0.0003	43.3	
Test0.0004	50.8	
Test0.0005	60.2	
Test0.0007	76.8	
Test0.001	115 KB	Small
Test0.002	210	
Test0.003	318	
Test0.004	457	
Test0.005	567	
Test0.007	817	
Test0.01	1.128 MB	
Test0.02	2.275	
Test0.03	3.402	
Test0.04	4.616	
Test0.05	5.601	
Test0.07	7.954	
Test0.1	11.325 MB	Large
Test0.2	22.817	
Test0.3	34.048	
Test0.4	45.307	
Test0.5	56.290	
Test0.7	79.702	
Test1.0	113.061 MB	
Test2.0	226.788	
Test3.0	340.717	
Test4.0	454.498	
Test5.0	568.096	
Test7.0	795.548	
Test10.0	1.111 GB	
Test20.0	2.225	
Test50.0	5.576	
Test100.0	11.141	

Figure 44 Names and Appropriate Sizes of Used XMark Documents

The Parsing testing group consists of the following testing scenarios:

- DOM processing (Par_DOM)
- Stream processing (Par_STREAM)

Each test case in both testing scenarios takes one XMark document from Figure 44 and parses it using the parsing method according to the governing testing scenario. For example, the test case `test0.0` belonging to a DOM Parsing testing scenario involves DOM parsing of an XMark document `test0.0.xml`.

However, not all XMark documents in Figure 44 are used for DOM parsing, because DOM memory requirements are increasing with the growing sizes of the XML documents. On the other hand, Stream processing has constant memory requirements, and, therefore, even the very large files can be processed using Stream approach.

Other Sources of XML Data

XMark is inevitably not the only generator of synthetic XML data, there are many others [Mly]. Arguments for choosing XMark are overviewed in Chapter 5.2.1

Instead of using XML generators, we can exploit a fixed set of XML data, such as exports of The Internet Movie Database, protein sequences, NASA astronomical data etc. However, these sources are often badly scalable, with many similar XML documents. What is more, since these sets of data represent only a subset of possible documents of a particular area, the advantage of true real data in comparison with XMark synthetic data is questionable.

5.2.2. Processing Rule Definition

The following subchapters describe the processing rules in the SW and HW environments.

The SW Environment

In `Par_STREAM` testing scenario, the processing rule consists of one action - Stream Parsing. Figure 45 denotes the processing rule together with the defined input and output contexts.

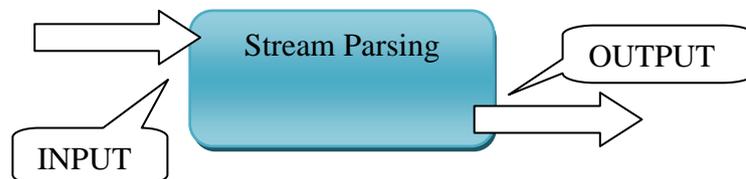


Figure 45 Processing Rule for `Par_STREAM` Testing Scenario

A processing rule for the `Par_DOM` testing scenario involves two actions - one for building a DOM tree in a memory and the second one for serializing the DOM tree back to the `OUTPUT` context (see Figure 46).

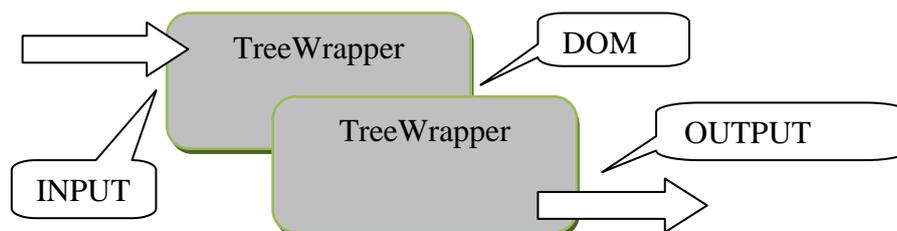


Figure 46 Processing Rule for `Par_DOM` Testing Scenario

HW Environment

In the HW environment, the processing rules for Par_DOM and Par_STREAM testing scenarios are the same (see Figure 47). Both processing rules contain the Matching action (see Chapter 4.5.5) and Transforming action with `identity.xsl` stylesheet copying an input document from INPUT to OUTPUT context. Transforming action must be used, because XI50 appliance does not decouple “Just Parsing” action. We can use a processing rule containing Matching and Results actions (see Chapter 4.5.5), however, only for Par_DOM testing scenario, because the Results action always buffers the XML document, which is definitely not acceptable for Par_STREAM testing scenario.



Figure 47 Processing Rule for Stream Parsing

Figure 48 denotes details of the Transforming action.

Figure 48 Detail of the Transforming Action

Input and Output sections denotes input and output contexts and are set to INPUT and OUTPUT. The Options section specifies `identity.xsl` as an XSL stylesheet used by the Transforming action. The meaning of the other settings can be found on [W4].

The settings in XML Manager (see Chapter 4.5.3) influence whether the DOM or Stream processing is used (and thus the Par_DOM or Par_STREAM testing scenario is executed). By default, all XSL transformations wait for parsing the whole input XML document, only then the XSL transformation is applied and the output is created. Hence, we have to enable stream processing for Par_STREAM testing scenario in XML Manager settings (see Chapter 7.1.2).

5.2.1. Tested Engines in the SW Environment

In the HW environment, we have no choice. In the SW environment, the most common parsers are

- Apache Xerces2 2.8.0 [W42] - Fully compliant with JAXP 1.3, DOM Level 3 and SAX 2.0.2.
- Apache Crimson 1.1 [W43] - This parser was a default one for Java 1.4, however, it supports only JAXP 1.1 (except for transformations) and DOM Level 2. The Apache Xerces2 project is preferred and further developed instead of Crimson.
- MXP1 - XML Pull Parser 3rd Edition [W44] - Based on different API specified on [W45].

For parsing XML data, Apache Xerces2 (2.8.0) is used in all testing scenarios in the SW environment. It is a default parser for Java 1.5, fully compliant with JAXP 1.3 specification and latest DOM and SAX specifications, and it has no reasonable competitors.

5.3. Testing Group: Validating XML Data

This testing group involves scenarios comprising of Validating actions.

5.3.1. Testing Data and Testing Hierarchy

XSDBench [W32] is an open-source XML Schema benchmark. However, it is not complete at all and does not contain support for XML Schema validation in Java. No other reasonable benchmarks can be found.

Nevertheless, the idea of XSDBench can be partially utilized in our own testing group. XSDBench differentiates two areas of using XSD file. To define:

- structure of an XML document
- data types of an XML document

XSD file describing only structure of an XML document should contain only build-in data types (such as data type “string”). On the other hand, XSD file holding data types of a described XML document should involve only various definitions of data types. In a real world, each XSD file typically contains both, definition of a structure as well as custom data types of characterized XML document.

In our testing group, we start with a base schema file, and then, add and remove complexity to/from this file and observe the performance gains or losses. The expressing power of the base schema file is on the level of a DTD schema file. We use the following levels of complexity of the schema files:

- DTD schema (Val_DTD_BASE testing scenario)
- XML Schema with expressing power of DTD - We use dtd2xs tool [W70] for converting a DTD to an XSD file. The XSD file then contains only structure definition on the level of DTD, string data types and some integrity restriction, which can be specified in DTD, such as ID and IDREF attributes. (Val_XSD_BASE)
- XML Schema with expressing power of DTD (see the Val_XSD_BASE), plus ID and IDREFs are replaced by simple strings (Val_XSD_REDUCED)
- XML Schema as in Val_XSD_BASE testing scenario, with complex data types containing pattern matching, enumeration values and other restrictions. (Val_XSD_EXTENDED)

As a base DTD file, `auction.dtd` file describing XML documents generated by XMark is used. This file has a potential for complex pattern matching and the results can be compared with the Parsing testing group results. As the source files, the same collection of testing files as in Parsing testing group is used (see 5.2.1).

Each level of complexity corresponds to one testing scenario. Each testing scenario has its own set of test cases, where each test case introduces a validation of one XMark file against the schema file determined by the testing scenario.

5.3.1. Processing Rule Definition

The following subchapters describe the processing rules in the SW and HW environments.

The SW Environment

In all scenarios in the SW environment, the processing rules consist of one action - XML Schema or DTD validation. The main difference between DTD and XML Schema validation concerns the fact that JAXP 1.3 specification does not allow decoupling of DTD validation. This means, DTD validation cannot be used in a processing rule of more actions unless it is used as a first action because, otherwise, the processed input XML document should have been parsed twice - once at the beginning of the processing rule and again as a part of DTD validation.

Figure 49 shows definition of the processing rule for XML Schema validation.



Figure 49 Processing Rule for Val_XSD_BASE Testing Scenario

HW Environment

In the HW environment, the processing rule is much more intuitive in comparison with Parsing testing group in Subsection 5.2. Figure 50 shows a processing rule for XML Schema validation testing scenarios. As we can see, the processing rule consists of two actions, the first one is the Matching action described already in Chapter 4.5.5 and the second action is a Validating action.



Figure 50 Processing Rule for Validating Action

Figure 51 shows the detail of the Validating action of the Val_XSD_BASE testing scenario.

Figure 51 Detail of Validating Action

The input and output contexts of the Validating action are set to INPUT and OUTPUT, because it is the only action in the processing rule (Matching action is not counted).

The middle part of Figure 51 defines several ways in which the incoming XML document can be validated. We have chosen the first variant - Validate Document via Schema URL, which means that all incoming XML documents are validated against the fixed XSD file and, therefore, XI50 can use XG4 Plug-in card for further acceleration of the validation process in a more efficient way. Other variants involve (apart from other less important choices) validation against XSD file specified in the incoming XML document and validation of XML documents with encrypted parts (see Section 5.5).

The Validating action is similar for other testing scenarios concerning XSD schema file, only the governed XSD file differs. However, DTD validations are not supported on XI50 appliance, thus, the testing scenario containing DTD validation is not performed on XI50.

5.3.1. Tested Engines in the SW Environment

The situation on the field of the tested engines is the same as in the Parsing testing group in the SW environment, because parsers are by design capable of validation process. Therefore, the Apache Xerces2 2.8.0 is used as JAXP 1.3 fully compliant parser for validating XML documents.

5.4. Testing Group: Transforming XML Data

This testing group covers XSL transformations.

5.4.1. Testing Data

XSLTMark is a benchmark for comprehensive measurement of performance of XSLT processors [W31]. The DataPower Company developed it in year 2000 for testing

performance of early versions of XI50 hardware appliances and compared results with accessible software solutions. In May 2005, IBM acquired DataPower Company and hardware appliances were further improved and integrated to IBM WebSphere platform.

Our testing scenarios contain the test cases involved in the original XSLTMark, but embedded to our testing framework. The original version of benchmark is currently hardly available and does not involve up-to-date drivers. However, although the XSLTMark is obsolete, utilizing its XSL stylesheets gives us a chance to accomplish tests once more on a more advanced version of hardware appliance and a more powerful software solution.

XSLTMark contains about 40 XSL stylesheets covering different aspects of XSLT Recommendation (see Subchapter “Areas of XSLT Processing”). All stylesheets conform only to the XSLT W3C Recommendation version 1.0. Apart from some implemented features of XSLT 2.0, XI50 is still not fully XSLT 2.0 compliant, so this is unimportant drawback.

The original set of XSLTMark stylesheets comprises our base testing scenario - Tra_XSTMark. The subsequent three testing scenarios use a subset of XSTMark styleshets with larger input XML documents. The testing scenarios are as follows (the full list of used XSL stylesheets and input XML documents in all testing scenarios is shown in Subsection “List of Test Cases”):

- Standard XSLTMark (Tra_XSTMark)
- XSLTMark with 10 times larger input XML documents (Tra_XSTMark_L)
- XSLTMark with 100 times larger input XML data (Tra_XSTMark_XL)
- XSLTMark with 1000 times larger input XML data (Tra_XSTMark_XXL)

One test case of all testing scenarios corresponds to application of one XSL stylesheet from XSLTMark on an appropriate input XML document using particular XSLT engine.

XSLTMark is the only reasonable XSLT benchmark that we have managed to find, fortunately suitable for our measurement according to wide range of tested areas. It would be very tough to write such a complex set of XSL stylesheets. The little bit smaller and easier input document sizes of XSLMark stylesheets for nowadays computing capabilities are balanced in three extension scenarios.

Areas of XSLT Processing

We can differentiate four areas of XSLT processing:

- XSLT rules pattern matching and template instantiation (MATCH)
- XSLT control structures (e.g. `xsl:choose`) and parameter passing (CONTROL)
- XPath selection of node sets (SELECT)
- XPath library functions such as string and node set operations (FUNCTION)

If the impacted area cannot be specified, GENERAL area is used.

List of Test Cases

The following list depicts test cases used in our testing scenarios. Each item of the following list contains a name of the test case, the character of an input XML document, a stylesheet description, and XSLT processing area covered by the test case. If the input XML document is used in more testing scenarios, it contains more versions of the input file (e.g. 100/1000 rows of a database table), the second value is used for Tra_XSTMark_L, the third value for

Tra_XSTMark_XL, and the fourth value for Tra_XSTMark_XXL testing scenarios. The test cases used in all four testing scenarios have bolded names. The names of the output XML documents are similar to the names of the input XML documents.

- **Alphabetize**
 - Input: `alphabetize.xml` - 100/1000/10000/100000 rows of database table
 - Style sheet: `alphabetize.xsl` - Sorts the elements in the input tree according to the names of elements
 - Impacted area of XSLT processing: SELECT, CONTROL
- **Attests**
 - Input: `attsets.xml` - Sales report
 - Style sheet: `attests.xsl` - Tests node-copying using named attribute sets
 - Impacted area of XSLT processing: GENERAL
- **Avts**
 - Input: `avts.xml` - 100/1000/10000/100000 rows of database table
 - Style sheet: `avts.xsl` - Tests expansion of attributes
 - Impacted area of XSLT processing: SELECT
- **Axis**
 - Input: Proprietary `axis.xml` file
 - Style sheet: `axis.xsl` - Tests XPath selection along the different axes
 - Impacted area of XSLT processing: SELECT
- **Backwards**
 - Input: Proprietary `backwards.xml` with tournament results
 - Style sheet: `backwards.xsl` - Reverses an order of elements in the input XML document
 - Impacted area of XSLT processing: CONTROL
- **Bottles**
 - Input: `bottles.xml` - Holds an auxiliary element with the initial size parameter containing number of bottles generated
 - Style sheet: `bottles.xsl` - Generates verses of the "99/999/9999/99999 bottles of beer on the wall" song
 - Impacted area of XSLT processing: FUNCTION, CONTROL
- **Breadth**
 - Input: `breadth.xml` - Broad and shallow XML document
 - Style sheet: `breadth.xsl` - Performs a search for a unique element in a large shallow tree
 - Impacted area of XSLT processing: SELECT, CONTROL
- **Brutal**
 - Input: `brutal.xml` - Inventory of items
 - Style sheet: `brutal.xsl` - Executes many XPath sum and count functions
 - Impacted area of XSLT processing: SELECT, FUNCTION, CONTROL
- **Chart**
 - Input: `chart.xml` - Sales report
 - Style sheet: `chart.xsl` - Generates HTML chart of sales data
 - Impacted area of XSLT processing: SELECT, CONTROL
- **Creation**
 - Input: `creation.xml` - 100/1000/10000/100000 rows of database table
 - Style sheet: `creation.xsl` - Tests `xsl:element` and `xsl:attribute`

- Impacted area of XSLT processing: GENERAL
- **Current**
 - Input: Proprietary `current.xml`
 - Style sheet: `current.xsl` - Tests complex XPath node selection
 - Impacted area of XSLT processing: SELECT
- **Dbonerow**
 - Input: `dbonerow.xml` - 10000 rows of database table
 - Style sheet: `dbonerow.xsl` - Selects a single row from a very large table
 - Impacted area of XSLT processing: SELECT, CONTROL
- **Dbtail**
 - Input: `dbtail.xml` - 100/1000/10000/100000 rows of database table
 - Style sheet: `dbtail.xsl` - Prints a table by traversing the following-sibling axis
 - Impacted area of XSLT processing: SELECT
- **Decoy**
 - Input: `decoy.xml` - 100/1000/10000/100000 rows of database table
 - Style sheet: `decoy.xsl` - Same stylesheet as in Patterns test case, with some decoy templates thrown
 - Impacted area of XSLT processing: MATCH
- **Depth**
 - Input: Proprietary `depth.xml`
 - Style sheet: `depth.xsl` - Performs a search for a unique element in a large tree
 - Impacted area of XSLT processing: SELECT, CONTROL
- **Encrypt**
 - Input: `encrypt.xml` - 100/1000/10000/100000 rows of database table
 - Style sheet: `encrypt.xsl` - Performs a Rot-13 substitution cipher on all element names and text nodes (it replaces each letter by another letter 13 characters further along the alphabet)
 - Impacted area of XSLT processing: FUNCTION
- **Functions**
 - Input: `functions.xml` - 100/1000/10000/100000 rows of database table
 - Style sheet: `functions.xsl` - Tests a variety of number and string functions
 - Impacted area of XSLT processing: FUNCTION
- **Game**
 - Input: Proprietary `game.xml`
 - Style sheet: `game.xsl` - Produces HTML table of the baseball game stats
 - Impacted area of XSLT processing: SELECT, FUNCTION, CONTROL
- **HTML**
 - Input: Proprietary `html.xml`
 - Style sheet: `html.xsl` - Produces HTML table with results of economy result of several divisions
 - Impacted area of XSLT processing: SELECT, CONTROL
- **Identity**
 - Input: `identity.xml` - 1000/10000/100000/1000000 rows of database table
 - Style sheet: `identity.xsl` - The identity transformation
 - Impacted area of XSLT processing: CONTROL
- **Inventory**
 - Input: Proprietary `inventory.xml`
 - Style sheet: `inventory.xsl` - Produces HTML table of the data

- Impacted area of XSLT processing: SELECT, CONTROL
- **Metric**
 - Input: `metric.xml` - Data in metric notation
 - Style sheet: `metric.xsl` - Converts metric units to English units
 - Impacted area of XSLT processing: FUNCTION
- **Number**
 - Input: Proprietary `number.xml`
 - Style sheet: `number.xsl` - Tests `format-number()` function
 - Impacted area of XSLT processing: FUNCTION
- **Oddtemplate**
 - Input: Proprietary `oddtemplate.xml`
 - Style sheet: `oddtemplate.xsl` - Tests a variety of complex match patterns
 - Impacted area of XSLT processing: MATCH, SELECT
- **Patterns**
 - Input: `patterns.xml` - 100/1000/10000/100000 rows of database table
 - Style sheet: `patterns.xsl` - Stylesheet contains extremely simple templates with tough patterns
 - Impacted area of XSLT processing: MATCH
- **Prettyprint**
 - Input: `prettyprint.xml` - 100/1000/10000/100000 rows of database table
 - Style sheet: `prettyprint.xsl` - Formats the input in legal HTML
 - Impacted area of XSLT processing: CONTROL, FUNCTION
- **Priority**
 - Input: Proprietary `priority.xml`
 - Style sheet: `priority.xsl` - Pops the first element off a priority Queue and returns the queue
 - Impacted area of XSLT processing: SELECT, CONTROL
- **Products**
 - Input: Proprietary `products.xml`
 - Style sheet: `products.xsl` - Produces HTML table from the product data
 - Impacted area of XSLT processing: SELECT, CONTROL
- **Queens**
 - Input: `queens.xml` - 6/8/10/12
 - Style sheet: `queens.xsl` - Solves the "Queens" problem
 - Impacted area of XSLT processing: FUNCTION, CONTROL
- **Reverser**
 - Input: `reverser.xml` - The Gettysburg Address - the most famous speech of Abraham Lincoln
 - Style sheet: `reverser.xsl` - Stylesheet copies the input document with text-node strings reversed
 - Impacted area of XSLT processing: FUNCTION, CONTROL
- **Stringsort**
 - Input: `stringsort.xml` - 1000/10000/100000/1000000 rows of database table
 - Style sheet: `stringsort.xsl` - Performs a sort based on string keys
 - Impacted area of XSLT processing: CONTROL
- **Summarize**
 - Input: `summarize.xml` - "Queens" stylesheet
 - Style sheet: `summarize.xsl` - Reports information about an XSL stylesheet

- Impacted area of XSLT processing: FUNCTION
- Total
 - Input: `total.xml` - Sales report
 - Style sheet: `total.xsl` - Reports on sales data
 - Impacted area of XSLT processing: SELECT, FUNCTION
- Tower
 - Input: `tower.xml` - 10/12/15/20 pieces of Hanoi tower building blocks
 - Style sheet: `tower.xsl` - Solves the “Towers of Hanoi” problem
 - Impacted area of XSLT processing: CONTROL, FUNCTION
- Trend
 - Input: `trend.xml` - Numerical data
 - Style sheet: `trend.xsl` - Computes trends in the input data
 - Impacted area of XSLT processing: SELECT, FUNCTION
- Union
 - Input: Proprietary `union.xml`
 - Style sheet: `union.xsl` - Performs complex pattern matching
 - Impacted area of XSLT processing: MATCH, SELECT
- XPath
 - Input: Proprietary `xpath.xml`
 - Style sheet: `xpath.xsl` - Performs another complex pattern matching
 - Impacted area of XSLT processing: MATCH

5.4.1. Processing Rule Definition

The following subchapters describe the processing rules in the SW and HW environments.

The SW environment

In all testing scenarios, the test cases involve one action - XSL transformation. Figure 52 depicts the definition of a processing rule with one XSL Transforming action executed on SAXON engine (see Chapter 5.4.2) with INPUT and OUTPUT contexts as input and output contexts.



Figure 52 Processing Rule for XSLTMark Test Cases in SW

A similar processing rule is defined for each combination of XSL stylesheet, XSLT engine and testing scenario.

HW Environment

A processing rule is very simple, it contains only Marching and Transforming actions (see Figure 53).

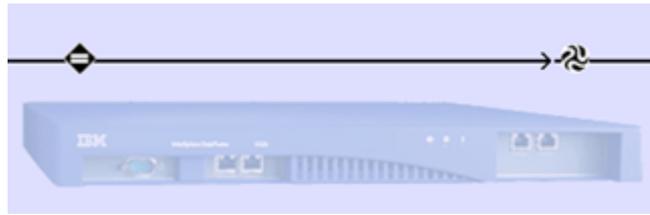


Figure 53 Processing Rule for XSLTMark Test Cases in HW

The detail of the Transforming action for a sample test case using `alphabetize.xml` stylesheet is revealed in Figure 54. Again, it contains INPUT and OUTPUT as `input` and `output` contexts. The middle part of Figure 54 defines which XSL stylesheet is used.

Figure 54 Detail of Transforming Action

As in Validating testing group, the XSL stylesheet is fixed for all documents using this processing rule, so that the XSL stylesheet can be pre-fetched and cached by XI50 appliance. In contrary to XML Schema validation, the XSL stylesheet can be specified in an incoming XML document as well, however, we must be careful, because in this case, an XSL stylesheet is cached only after the first use of it.

5.4.2. Tested Engines in the SW Environment

In the HW environment, the XSLT engine is obvious. In the SW environment, the list of available engines (among others) involves:

- XALAN 2.7.1
- SAXON-B 9.0.0.2
- XSLTC

Standard XSLT engine in Java 1.5 is XSLTC. XSLTC is based on XALAN engine, however, uses compiled stylesheets.

We use all these XSLT engines, so that we can compare them and stand the strongest one against XI50 in the “Onion” testing suite (Section 6).

Character of some XSL stylesheets enables streaming processing of XSL stylesheets. However, no used XSLT engine in the SW environment is capable of streaming. SAXON-A (higher version of SAXON engine) should have some restricted possibility to mark a part of an XSL stylesheet as streamable, however, this task must be prepared manually for all XSL stylesheets and what is more, SAXON-A is not freely available and we do not have chance to try it.

5.5. Testing Group: Securing XML Data

The goal of this testing group is to measure the performance of digital signing and encrypting of XML documents according to W3C standards XML Signature and XML Encryption (see Chapter 2.1.4).

5.5.1. Testing Data and Testing Hierarchy

Since there exists no special benchmark for testing only encryption and/or signing of XML documents, the problem of choosing right data for the Securing testing group is related to choosing an appropriate generator of XML documents. We use the same generator as for Parsing testing group, so that we can straightforwardly compare the measures gathered in the Parsing and Securing testing group.

The encrypting and decrypting testing scenarios are following:

- Encrypting the whole incoming XMark document using asymmetric RSA cipher with 2048 bits long key (RSA2) for encrypting the actual symmetric key used for the encryption. The symmetric algorithm is 3DES in Sec_ENC_ENC_RSA2_3DES testing scenario and AES CBC with 128 and 256 bits long key in Sec_ENC_ENC_RSA2_AES128 and Sec_ENC_ENC_RSA2_AES256 testing scenarios. RSA algorithm with 1024 bits long key is examined in the HW environment as well in Sec_ENC_ENC_RSA_3DES and Sec_ENC_ENC_RSA_AES256 testing scenarios.
- Decrypting the whole XML document encrypted using RSA2 and AES 256 CBC or 3DES (Sec_ENC_DEC_RSA2_AES256 and Sec_ENC_DEC_RSA2_3DES).
- Encrypting one element which occurs repeatedly in an XML document (such as an element holding credit card numbers of persons participating on stock market business) with RSA2 and AES 256. RSA2 is chosen for higher security and AES 256 for quicker encryption and higher security in comparison with 3DES (Sec_ENC_ENC_PART_RSA2_AES256).

Moreover, the signing and verifying testing scenarios are as follows:

- Signing the whole document using asymmetric RSA cipher with 1024 bits long key (RSA), RSA2 and DSA with 1024 bits long key (DSA) together with SHA1 digest algorithm. Therefore, we have three testing scenarios - Sec_SIG_SIG_DSA_SHA1, Sec_SIG_SIG_RSA_SHA1, and Sec_SIG_SIG_RSA2_SHA1.
- Verifying the whole document signed by DSA and SHA1 (the testing scenario Sec_SIG_VER_DSA_SHA1)

5.5.2. Processing Rule Definition

The following subchapters describe the processing rules in the SW and HW environments.

SW Environment

All scenarios consists of one Encrypting/Decrypting or Signing/Verifying action using a defined combination of cryptographic algorithms and two TreeWrapper actions for building a DOM tree of an input XML document and for serializing encrypted/decrypted/signed XML document to an output stream (Encrypting, Decrypting, Signing, and Verifying actions must operate with the whole XML document). Figure 55 demonstrates the processing rule definition for the testing scenario signing the whole XMark document using DSA and SHA1 algorithms (testing scenario Sec_SIG_VER_DSA_SHA1).

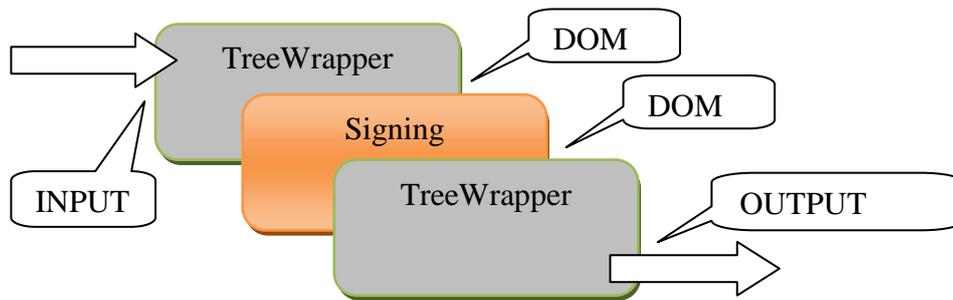


Figure 55 Signing Processing Rule for DSA and SHA1 Algorithms

Processing rules containing the Encrypting or Decrypting actions instead of Signing action are very similar. The Verifying action has a null output and, therefore, the second TreeWrapper action uses the output of the first TreeWrapper action as an input (see Figure 56)

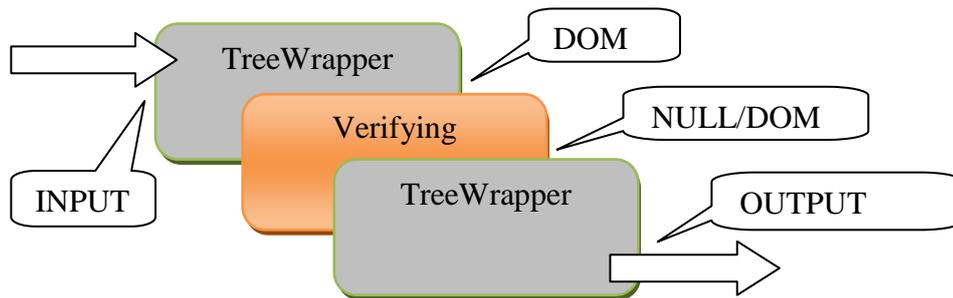


Figure 56 Verifying Processing Rule for DSA and SHA1 Algorithms

HW Environment

The processing rules of all testing scenarios in the Securing testing group contain two actions, the Matching action and Encrypting, Decrypting, Signing, or Verifying action, depending on the particular testing scenario.

Figure 57 depicts the processing rule used in the Sec_ENC_ENC_RSA2_AES256 testing scenario.



Figure 57 Encrypted using RSA2 and AES 256

Figure 58 shows the detail of the Encrypting action. As in other testing groups of the “Flat” testing suite, it contains INPUT and OUTPUT as input and output contexts. The middle part of Figure 58 specifies that we use Standard XML Encryption, the input to this action is the whole Raw XML document, the used certificate is testRSA2_Cer (based on RSA2 algorithm), and the symmetric encryption algorithm is AES256-CBC. Moreover, the encryption type is Element which means that the whole desired element is encrypted, not just the content of the element. Meaning of the other settings can be found on [W4] and [W19].

The screenshot displays the configuration for an 'Encrypt' action. The 'Input' section is set to 'INPUT'. The 'Options' section includes the following settings:

- Action Type:** Encrypt
- Envelope Method:** Standard XML Encryption
- Message Type:** Raw XML Document
- Processing Control File:** store:///encrypt.xsl
- Output Type:** Default
- Asynchronous:** off
- Use Dynamically Configured Recipient Certificate:** off
- Recipient Certificate:** testRSA2_Cer
- Symmetric Encryption Algorithm:** AES256-CBC
- Key Transport Algorithm:** rsa-pkcs1
- Encryption type:** Element
- Use SAML Encryption for SAML Elements:** off

The 'Output' section is set to 'OUTPUT'.

Figure 58 Detail of Encrypting Using RSA2 and AES 256

Figure 59 depicts actions of the processing rule involved in the testing scenario Sec_SIG_SIG_DSA_SHA1.



Figure 59 Signing Using DSA and SHA1 Algorithms

Detailed information about the Signing action is depicted in Figure 60. The Signing action involves the INPUT and OUTPUT as input and output contexts. The other settings particularize that the Enveloped Method is used to sign the whole Raw XML document. The selected enveloped method has an impact on the location and syntax of the element containing signature metadata in the resulting XML document. Furthermore, `testDSA_PKey` is used as a private key for signing the input XML document digested by `sha1` message digest algorithm. The certificate `testDSA_Cer` (s public key to a `testDSA_PKey`) is attached to the signature metadata in the resulting XML document. Meaning of the other settings can be found on [W4] and [W16].

Input	
Input	INPUT <input type="text"/> INPUT <input type="button" value="v"/>
Options	
<input checked="" type="radio"/> Sign	
Action Type	Sign <input type="button" value="v"/> *
Envelope Method	<input checked="" type="radio"/> Enveloped Method <input type="radio"/> Enveloping Method <input type="radio"/> SOAPSec Method <input type="radio"/> WSSec Method <input type="radio"/> Advanced *
Message Type	<input type="radio"/> SOAP Message <input type="radio"/> SOAP With Attachments <input checked="" type="radio"/> Raw XML Document <input type="radio"/> Selected Elements (Field-Level) <input type="radio"/> Advanced *
Processing Control File	<input type="text" value="store:///sign-enveloped.xsl"/> <input type="text" value="store:///"/> <input type="button" value="v"/> <input type="text" value="sign-enveloped.xsl"/> <input type="button" value="v"/> <input type="button" value="Upload..."/> <input type="button" value="Fetch..."/> <i>Stylesheet Summary: Generate an enveloped XML digital signature. The signature is added as the last child element.</i>
Output Type	Default <input type="button" value="v"/>
Asynchronous	<input type="radio"/> on <input checked="" type="radio"/> off
Key	testDSA_PKey <input type="button" value="v"/> + ... <input checked="" type="checkbox"/> Save
Certificate	testDSA_Cer <input type="button" value="v"/> + ... <input checked="" type="checkbox"/> Save
Signing Algorithm	dsa <input type="button" value="v"/> <input checked="" type="checkbox"/> Save
Canonicalization Algorithm	Exclusive <input type="button" value="v"/> <input type="checkbox"/> Save
Message Digest Algorithm	sha1 <input type="button" value="v"/> <input checked="" type="checkbox"/> Save
Key/Certificate Base Name	<input type="text"/> <input type="checkbox"/> Save
<input type="button" value="Add Parameter"/>	
Output	
Output	OUTPUT <input type="text"/> OUTPUT <input type="button" value="v"/>
<input type="button" value="Delete"/> <input type="button" value="Done"/> <input type="button" value="Cancel"/>	

Figure 60 Detail of Signing Using DSA and SHA1 Algorithms

The processing rules for decrypting encrypted XML documents and verifying signed XML documents are similar, only the chosen Encrypting and Signing actions are replaced by Decrypting and Verifying actions and in case of Verifying action, the output context is set to NULL.

5.5.3. Tested Engines in the SW Environment

JSR 105 specification of unified Java interface for XML Signature was released in 2005 and we use its reference implementation of XML Signature.

On the other hand, JSR 106 specifications of unified Java interface for XML Encryption is still in Proposed Public Review Draft status, thus, the reference implementation is not yet

available. There are proprietary implementation of XML Encryption W3C standard, which unfortunately do not have a common interface:

- Package `com.ibm.ws.wssecurity.xss4j` (`xmlsecurity.jar`) - IBM library used by default in IBM WebSphere Application Server
- Package `org.apache.xml.security` [W47] - Apache XML Security project
- Package `org.bouncycastle` [W48] - Open source Cryptography API of Legion of the Bouncy Castle group

Unfortunately, Apache XML Security project has unavailable documentation and last update in May 2007. Cryptography API from Legion of the Bouncy Castle is suitable, however we have finally selected IBM library, which is a de facto Java standard for XML Encryption on IBM products. All above packages support digital signature as well, however, each package with its own proprietary interface.

6. DEFINING THE “ONION” TESTING SUITE

The “Onion” testing suite takes advantages of knowledge and experiences gained in the “Flat” one and involves the testing groups containing various combinations of actions over XML data mixed together in a single processing rule. In other words, the testing groups create progressions of testing scenarios, starting from a very simple one (with a processing rule with just one action) and ending with the most complex one (with many meaningful actions).

6.1. The Outline of the Testing Groups and Testing Scenarios

This testing suite consists of the following testing groups and its testing scenarios:

- Auction testing group - It creates a HTML report involving buyers on a stock market and auctioned items
 - Auction_XSLT
 - Auction_VAL_XSLT
 - Auction_VAL_XSLT_SIGN
 - Auction_VAL_XSLT_SIGN_ENC
- CSVOutput testing group - This testing group covers exporting comma separated values (CSV values) from an XML database of employees
 - CSV_XSLT
 - CSV_XSLT_XSLT2
 - CSV_XSLT_VAL_XSLT2
 - CSV_VER_XSLT3_XSLT_VAL_XSLT2
 - CSV_DEC_VER_XSLT3_XSLT_VAL_XSLT2

Testing groups in the “Onion” testing suite are composed so that they pretend some real world testing scenarios, such as generating HTML reports or exporting XML to CSV file. Besides, Auction testing group utilizes tough XSL stylesheet, the toughest version of XML Schema validation introduced in the Validating testing group and Encrypting and Signing actions from the Securing testing group. On the other hand, CSVOutput testing group contains easy streamable XSL transformations, XML Schema validation, and Verifying and Decrypting actions from the Securing testing group. XSLTC engine is used in all testing scenarios as the XSLT engine in the SW environment due to the best results in Transforming testing group.

6.2. Testing Group: Auction

Auction testing group covers the stress testing combining medium-to-tough Validating action, tough Transforming action, Signing action, and Encrypting action in its most complex testing scenario.

6.2.1. Testing Data and Testing Hierarchy

Auction testing group employs some of the XMark testing data used in the Parsing testing group. To be more precise, the XMark files depicted in Figure 61 are utilized.

Filename	Size (KBs, MBs)
Test0.002	210 KB
Test0.004	457 KB
Test0.005	567 KB
Test0.007	817 KB
Test0.01	1.128 MB

Figure 61 Names and Appropriate Sizes of Used XMark Documents in Auction

As we can see, the list of used XMark files is noticeably shorter than the list in Figure 44 and contains rather smaller files (because the Transforming action is relatively tough). On the other hand, each file is tested with different levels of concurrency (1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, and 100).

The testing scenarios of the Auction testing group are as follows:

- Auction_XSLT - The first scenario involves only one action in a processing rule - Transforming action using XSL stylesheet `auctionBuyers.xsl` which generates a HTML report involving buyers and its auctioned items. The transformation is chosen for its non-triviality and non-streamability.
- Auction_VAL_XSLT - It includes the Transforming action from the previous testing scenario and adds a Validating action utilizing `auctionExtended.xsd` XSD file introducing the toughest variant of XSD file from Validating testing group (see Subsection 5.3).
- Auction_VAL_XSLT_SIGN) - It uses the actions defined in Auction_VAL_XSLT testing scenario and adds Signing action of the whole XML document using SHA1 digest algorithm and DSA asymmetric cipher.
- Auction_VAL_XSLT_SIGN_ENC - The most complex testing scenario of the Auction testing group contains Transforming, Validating, and Signing actions from the previous testing scenarios and adds Encrypting action of the whole XML document using AES 256 symmetric cipher (with 256 bits long key) and RSA2 asymmetric cipher (with 2048 bits long key).

6.2.2. Processing Rule Definition (Auction_XSLT)

In both testing environments, the processing rule contains one Transforming action using `auctionBuyers.xsl` with input and output contexts set to INPUT and OUTPUT. Figure 62 depicts the processing rule in the SW and HW environments.



Figure 62 Processing Rule for Auction_XSLT Testing Scenario

6.2.3. Processing Rule Definition (Auction_VAL_XSLT)

A processing rule for the Auction_VAL_XSLT differs in both environments. In the SW environment, the processing rule is rather complicated. If the Validating action produces not-null output (output context is not set to NULL), it cannot directly process INPUT context (see restrictions of JAXP [W26]). Therefore, there are several solutions to substitute the expected PIPED context between Validating and Transforming action:

- Adding one extra auxiliary Transforming action before Validating action and leaving PIPED context between Validating and Transforming action.
- Setting output context of the Validating action to NULL and:
 - Building a DOM tree (using TreeWrapper action)
 - Copying an input stream of XML data to a buffer and re-reading it again by a Validating action (However, this solution bypasses the standard testing framework behaviour.)

We choose the second solution involving a creation of a DOM tree. The testing data are rather small, so the tree in memory does not represent a trouble (the largest testing file in this testing scenario has about 1 MB).

To sum up, the processing rule involves three actions - TreeWrapper action for building the DOM tree, Validating action using `auctionExtended.xsd`, and Transforming action using `auctionBuyers.xsl` stylesheet as in the Auction_XSLT testing scenario.

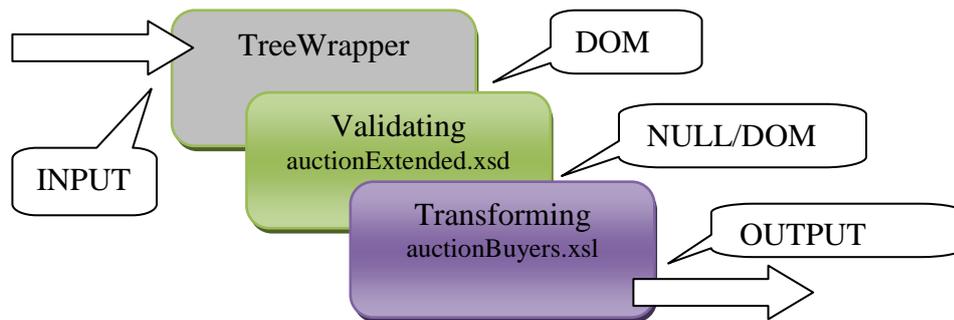


Figure 63 Processing Rule for Auction_VAL_XSLT Testing Scenario in SW

In the HW environment, there is no problem with PIPED context between Validating and Transforming action, however, in order to have as much comparable processing rules in both environments as possible, the processing rule contains Validating and Transforming actions using the same XSL and XSD files as in the SW environment and with the contexts as in Figure 64.

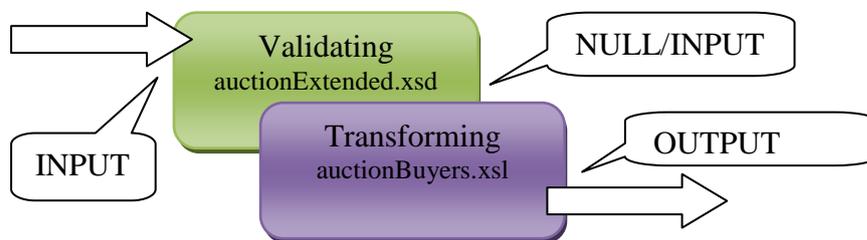


Figure 64 Processing Rule for Auction_VAL_XSLT Testing Scenario in HW

6.2.4. Processing Rule Definition (Auction_VAL_XSLT_SIGN)

The processing rules differ in the HW and SW environments. In the SW environment, this testing scenario contains TreeWrapper, Validating, and Transforming actions as in Auction_VAL_XSLT testing scenario and adds Signing and TreeWrapper actions for signing the result of the Transforming action and serializing it to the output stream. The contexts are as in Figure 65.

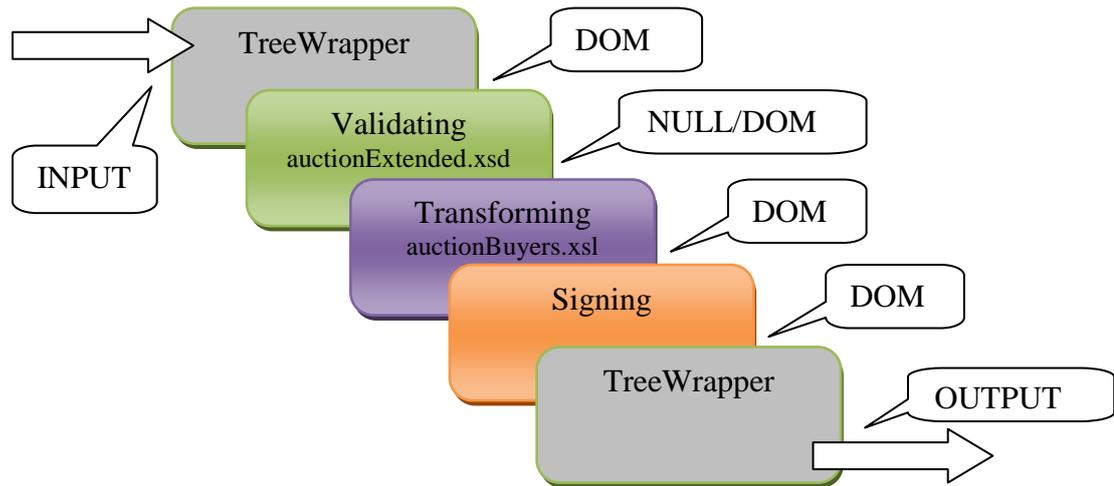


Figure 65 Processing Rule for Auction_VAL_XSLT_SIGN Testing Scenario in SW

In the HW environment, the testing scenario embraces the same actions as Auction_VAL_XSLT testing scenario and simply adds the Signing action and sets the contexts as in Figure 66.

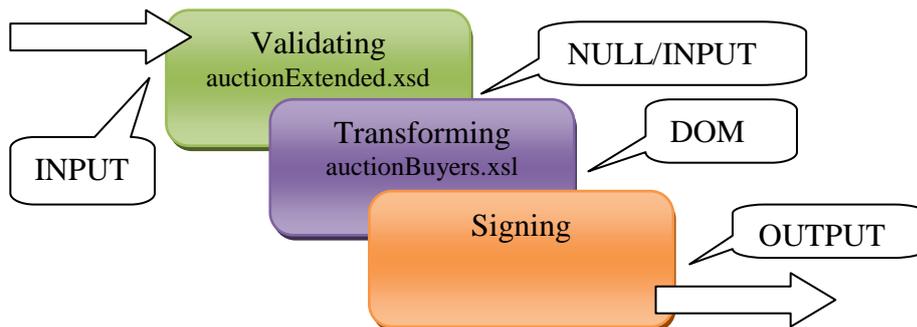


Figure 66 Processing Rule for Auction_VAL_XSLT_SIGN Testing Scenario in HW

6.2.5. Processing Rule Definition (Auction_VAL_XSLT_SIGN_ENC)

Again, the processing rules differs in HW and the SW environments. In the SW environment, the testing scenario contains actions as in the Auction_VAL_XSLT_SIGN testing scenario and adds Encrypting action between Signing and the second TreeWrapper actions, so that the signed XML document is encrypted and serialized to the output stream. The contexts are as in Figure 67.

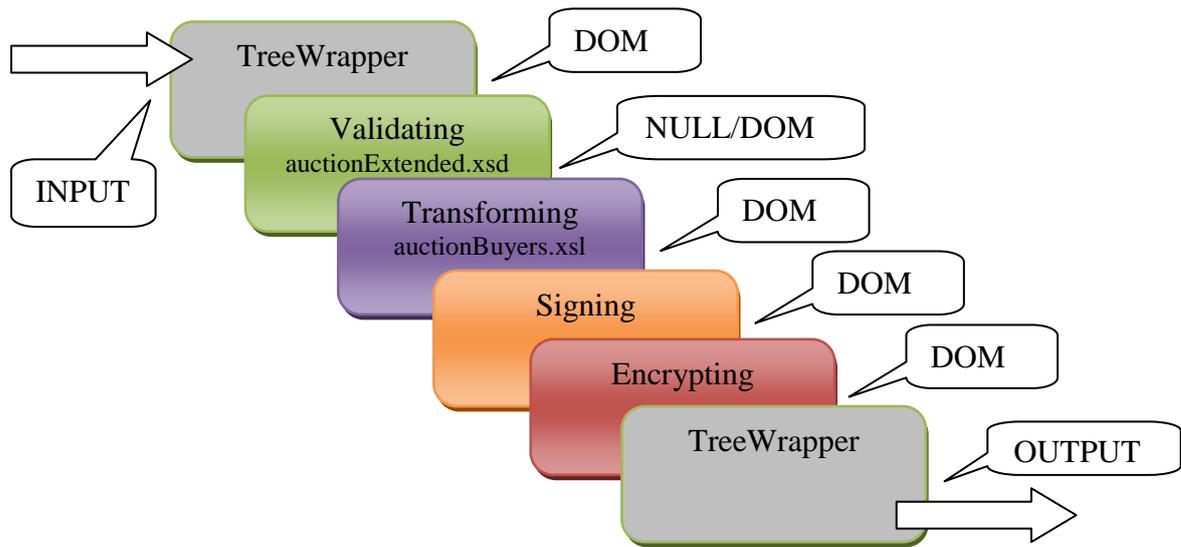


Figure 67 Rule for Auction_VAL_XSLT_SIGN_ENC Testing Scenario in SW

In the HW environment, the testing scenario Auction_VAL_XSLT_SIGN_ENC involves the same actions as Auction_VAL_XSLT_SIGN testing scenario, adds the Encrypting action at the end of the queue of actions, and uses contexts as in Figure 68.

Figure 66

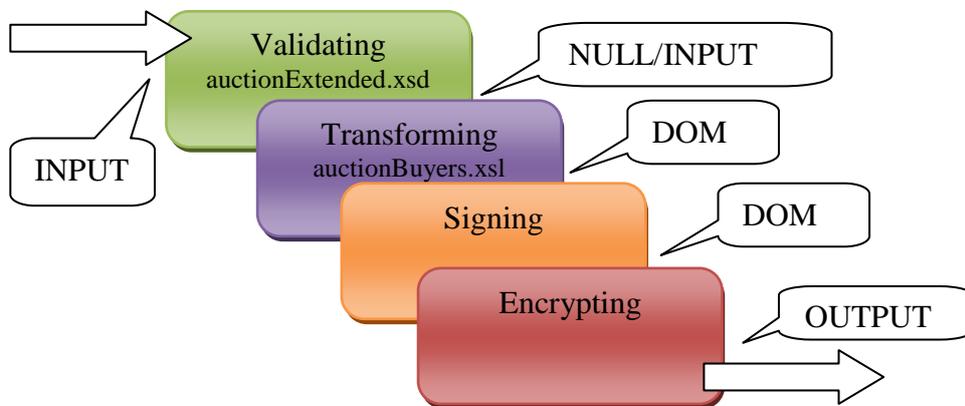


Figure 68 Rule for Auction_VAL_XSLT_SIGN_ENC Testing Scenario in HW

6.2.6. Tested Engines in the SW Environment

In the HW environment, the tested engines are obvious. In the SW environment, the Validating action uses XERCES2 as in the Parsing and Validating testing groups, Transforming action utilizes XSLTC engine.

6.3. Testing Group: CSVOutput

CSVOutput testing group generates in a secured way comma separated values (CSV values) from an XML database of employees. It involves stress testing combining three easy streamable XSL transformations, easy Validating, Verifying, and Decrypting actions in its most complex testing scenario.

6.3.1. Testing Data and Testing Hierarchy

The CSVOutput testing group embraces different sizes of testing data already used in the Transforming testing group. To be more precise, the files in Figure 69 contain successively 20, 200, 2000, and 20000 elements `row` depicted in Figure 70. Each file is tested with different levels of concurrency (1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, and 100).

Filename	Size (KBs, MBs)
Rows20	3.392 KB
Rows200	34.563 KB
Rows2000	356.829 KB
Rows20000	3.599 MB

Figure 69 Names and Appropriate Sizes of used XML Documents in CSVOutput

```
<?xml version="1.0" encoding="utf-8" ?>
<table>
  <row>
    <id>0000</id>
    <firstname>Al</firstname>
    <lastname>Aranow</lastname>
    <street>1 Any St.</street>
    <city>Anytown</city>
    <state>AL</state>
    <zip>22000</zip>
  </row>
  ...
</table>
```

Figure 70 Structure of the Database of Employees

The testing scenarios of the CSVOutput testing group are following:

- CSV_XSLT - The first testing scenario contains only one Transforming action in a processing rule, using `avtsEdited.xml` stylesheet for converting sub-elements of all elements `row` to attributes. The testing scenario is streamable in the HW environment.
- CSV_XSLT_XSLT2 - The processing rule of the second testing scenario involves the processing action of the CSV_XSLT testing scenario and the core XSL stylesheet `avtsToCSV.xml` exporting the output of the first Transforming action as comma separated values (CSV). The testing scenario is streamable in the HW environment. The stylesheet `avtsToCSV.xml` is inspired by [Man].
- CSV_XSLT_VAL_XSLT2 - This testing scenario adds Validating action governed by `avtsBeforeCSV.xsd` to the processing rule of the CSV_XSLT_XSLT2 testing scenario. The testing scenario is streamable in the HW environment.
- CSV_VER_XSLT3_XSLT_VAL_XSLT2 - The processing rule contains actions from the CSV_XSLT_VAL_XSLT2 testing scenario and appends Transforming and Verifying actions. Moreover, the input XML data slightly differ from the simpler testing scenarios of this testing group, because this testing scenario expects signed versions of the files in Figure 69 using DSA and SHA1 algorithms. These testing data are firstly verified by the Verifying action and after that, the Transforming action using `avtsStripSignature.xml` strips off the Signature element and sends the rest of the XML data to the Transforming action utilizing `avtsEdited.xml` stylesheet. The processing rule is non-streamable as a whole, because the Verifying action needs access to the whole XML document.

- **CSV_DEC_VER_XSLT3_XSLT_VAL_XSLT2** - The most complex testing scenario includes the processing actions from **CSV_VER_XSLT3_XSLT_VAL_XSLT2** testing scenario and adds a new Decrypting action for XML documents encrypted by AES 256 and RSA2 ciphers. Therefore, the input testing data must slightly differ from the files in Figure 69 and from files mentioned in the testing scenario **CSV_VER_XSLT3_XSLT_VAL_XSLT2**, so that the Decrypting action can be used. To be more precise, the testing data are firstly signed using DSA and SHA1 algorithms (as in the previous testing scenario), later on, encrypted using RSA2 and AES 256 ciphers and only after that, they can be used as testing data to this testing scenario. The processing rule is non-streamable as a whole, because the Verifying and Decrypting actions need access to the whole XML document.

6.3.2. Processing Rule Definition (CSV_XSLT)

In both testing environments, the processing rule involves one Transforming action using `avtsEdited.xsl` stylesheet with input and output contexts set to INPUT and OUTPUT. Figure 71 depicts the processing rule in the SW and HW environments.



Figure 71 Processing Rule for CSV_XSLT Testing Scenario

6.3.3. Processing Rule Definition (CSV_XSLT_XSLT2)

In both testing environment, the processing rule contains two Transforming actions PIPED together. The first one is utilizing `avtsEdited.xsl` stylesheet from the **CSV_XSLT** testing scenario and the second uses `avtsToCSV.xsl` key stylesheet. The contexts look like in Figure 72.

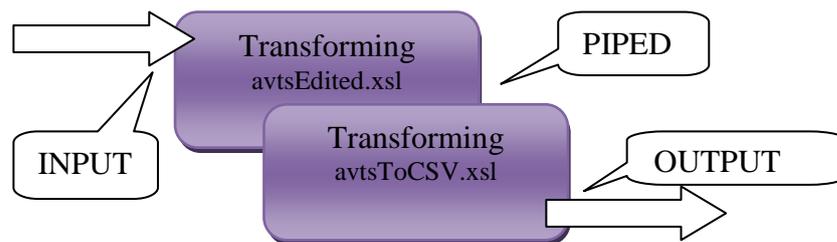


Figure 72 Processing Rule for CSV_XSLT_XSLT2 Testing Scenario

6.3.4. Processing Rule Definition (CSV_XSLT_VAL_XSLT2)

In both environments, the processing rules contain two Transforming actions as in **CSV_XSLT_XSLT2** testing scenario and add a Validating action. All actions in both processing rules are PIPED as in Figure 73.

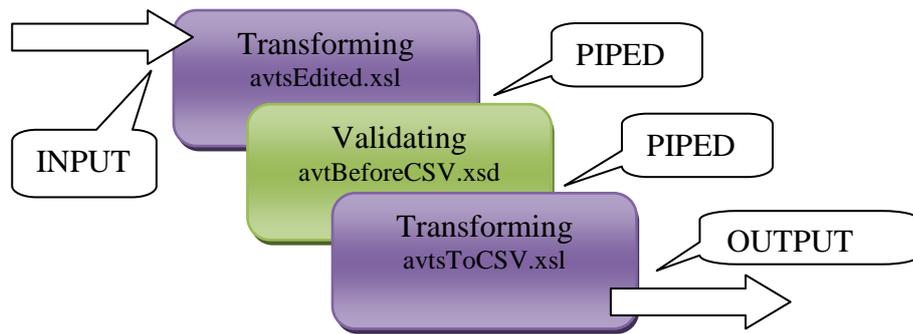


Figure 73 Processing Rule for CSV_XSLT_VAL_XSLT2 Testing Scenario

6.3.5. Processing Rule Definition (CSV_VER_XSLT3_XSLT_VAL_XSLT2)

The processing rules differ in the HW and SW environments. In the SW environment, the testing scenario contains the actions as in the CSV_XSLT_VAL_XSLT2 testing scenario and adds TreeWrapper, Verifying, and Transforming actions. The TreeWrapper action builds a DOM tree of the input document, subsequently, the Verifying action verifies the built DOM tree and later, the Transforming action takes the DOM tree created by the TreeWrapper action and strips off the Signature element and sends the XML data further. The contexts are as in Figure 74.

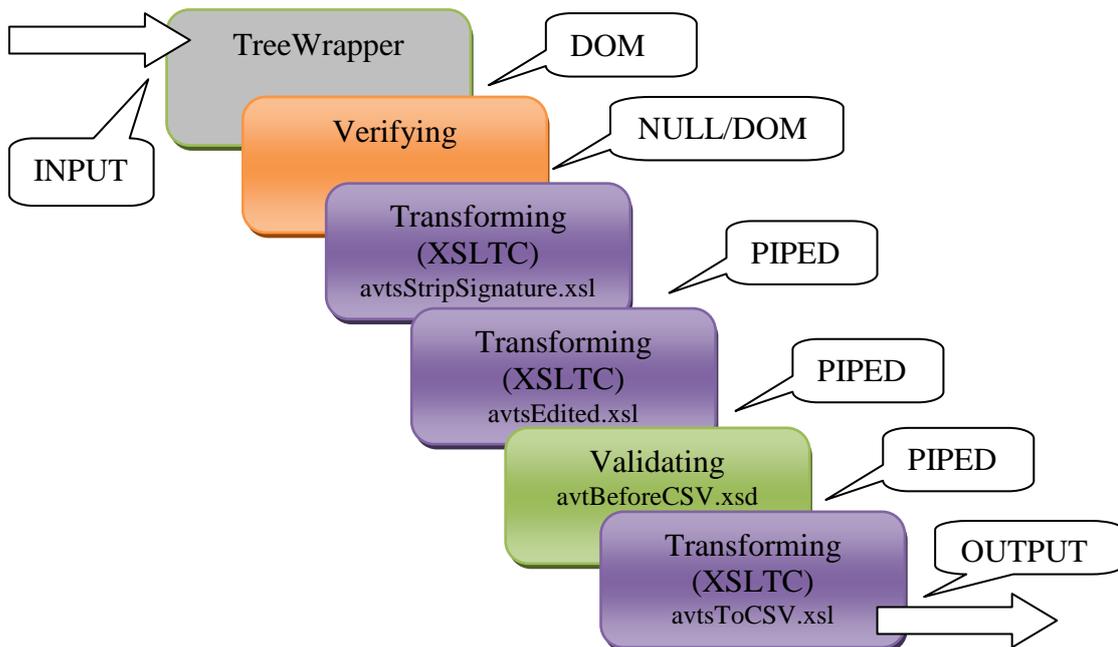


Figure 74 Rule for CSV_VER_XSLT3_XSLT_VAL_XSLT2 Testing Scenario in SW

In the HW environment, the actions are as in the CSV_XSLT_VAL_XSLT2 testing scenario, however, the Verifying and Transforming actions are added. The contexts are as in Figure 75.

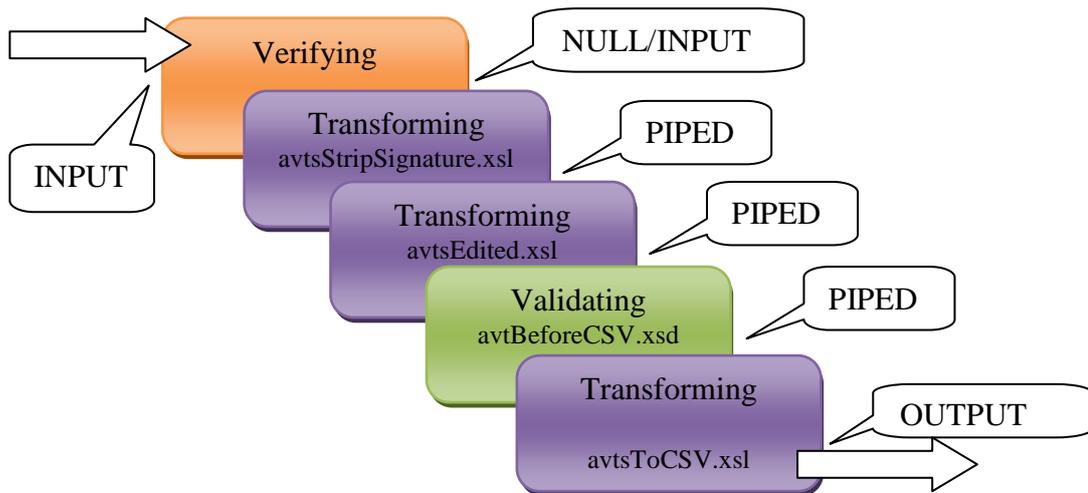


Figure 75 Rule for CSV_VER_XSLT3_XSLT_VAL_XSLT2 Testing Scenario in HW

6.3.6. Rule Definition (CSV_DEC_VER_XSLT3_XSLT_VAL_XSLT2)

The processing rules in both testing scenarios use the same actions as in CSV_VER_XSLT3_XSLT_VAL_XSLT2 testing scenarios of the respective environment and insert Decrypting action before the Verifying action. The contexts are as in Figure 76 for the SW environment and in Figure 77 for the HW environment.

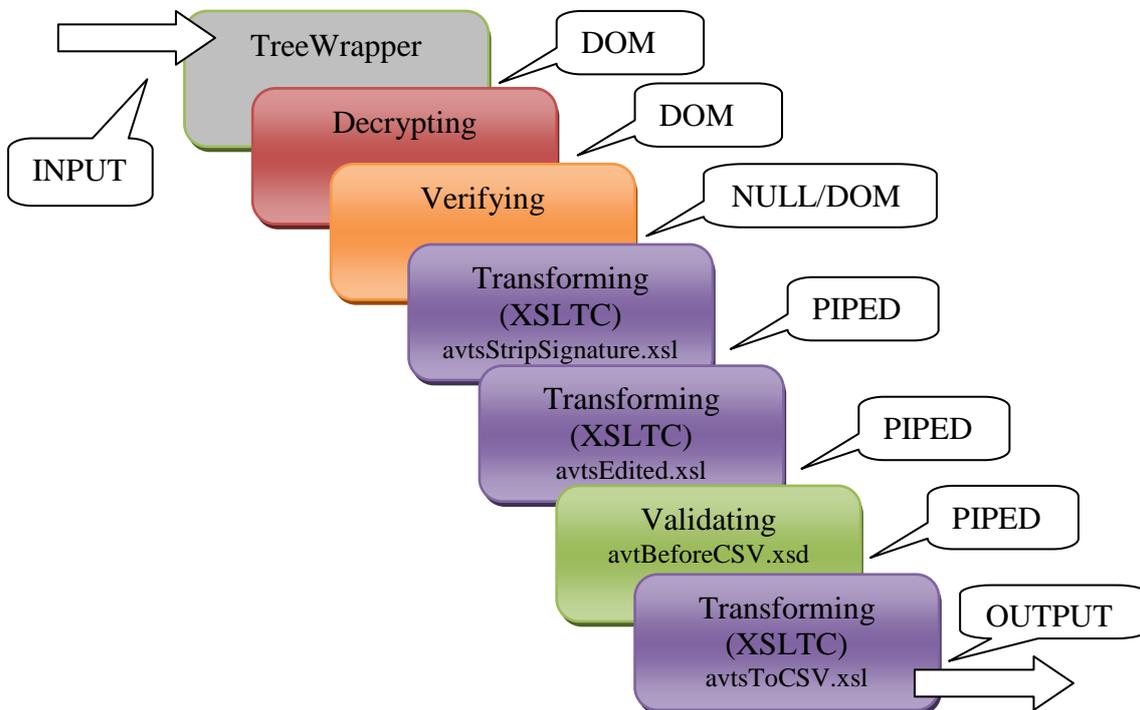


Figure 76 Rule for CSV_DEC_VER_XSLT3_XSLT_VAL_XSLT2 Scenario in SW

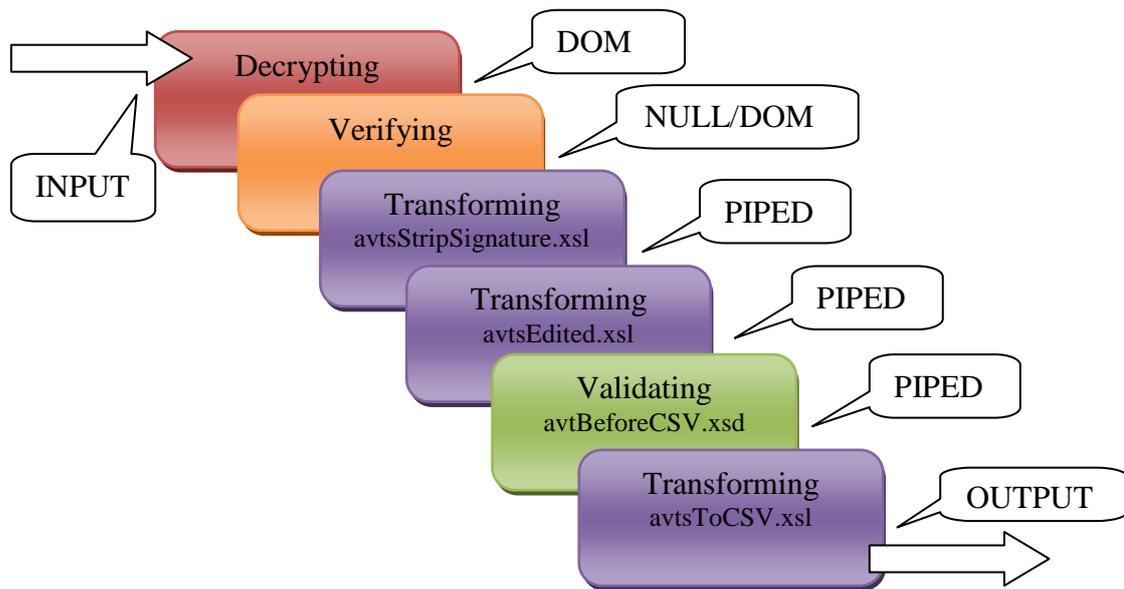


Figure 77 Rule for CSV_DEC_VER_XSLT3_XSLT_VAL_XSLT2 Scenario in HW

6.3.7. Tested Engines in the SW Environment

In the HW environment, the tested engine cannot be changed. In the SW environment, the Validating action uses XERCES2 as in the Parsing and Validating testing group and all Transforming actions utilize XSLTC engine.

IV. TESTING - TESTING AND COMPARING

7. ENVIRONMENT SETTINGS

This section summarize settings of both testing environments

7.1. The HW Environment

Firstly, let us have a look at settings of the HW environment. Chapter 7.1.1 holds configuration, Chapter 7.1.2 tuning possibilities, Chapter 7.1.3 monitoring, and Chapter 7.1.4 debugging capabilities of this environment.

7.1.1. Configuration

The configuration of the HW environment (XI50 appliance) is as follows:

- IBM WebSphere DataPower Integration Appliance XI50,
- XG4 add-on accelerator card installed
- Firmware version XI50.3.6.1.4 Build 156561cf

7.1.2. Tuning Possibilities

By default, XI50 appliance caches 5000 XML documents. The number of cached documents can be redefined within a range of 1 - 250.000 documents or the size of the caching memory can be specified. We set the caching memory to zero, so that no XML document is cached, because in a real-world scenario, two documents are rarely the same.

Furthermore, XI50 appliance caches 256 XSL stylesheets by default, the number of cached stylesheets can vary between five and 1.000.000. The default value (256 stylesheets) is sufficient, because our tests do not contain more than 50 XSL stylesheets. Therefore, all stylesheets in our tests are cached which corresponds to a real-world scenario, where the majority of regularly used XSL stylesheets should be cached.

Moreover, we disable all limits on the size of a single element in an XML document, the size of the whole XML document, and on the maximum depth of elements in an XML document.

By default, XML Schema validations are performed using SAX model, XSL transformations use DOM model. Alternatively, SAX processing can be used for transforming XML documents (if the nature of an XSL stylesheet enables it), which is called streaming. If the XSL stylesheet cannot be fully streamed, partial streaming is tried, which we can imagine as creation of small local DOM trees during the transformation. Streaming settings can be enabled or disabled in XML Manager settings at two levels:

- Stream, if not possible, raise error
- Stream, if not possible, process the input document in a standard way, which means building a DOM tree

When the streaming is enabled, it is emphasized in the definition of the testing scenario.

All the above tuning possibilities can be redefined in XML Manager settings of each XML Firewall.

7.1.3. Monitoring Capabilities

Under the main menu “Status” in web-based interface of XI50 appliance (see Figure 25), the following usable statistics can be obtained:

- CPU Usage - The usage of the general processing engine

10 sec	66	%
1 min	40	%
10 min	33	%
1 hour	19	%
1 day	19	%

Figure 78 CPU Usage (XI50)

- System Usage - The total workload (counted as load of the general processing engine and all other resources, such as accelerators, XG4 add-on card)

interval	1000	msec
load	86	%
work list	2	

Figure 79 System Usage (XI50)

- Memory Usage - Information about free, used and total memory usage

Memory Usage	16	%
Total Memory	3369836	kbytes
Used Memory	566152	kbytes
Free Memory	2803684	kbytes
Requested Memory	1651532	kbytes
XG4 Resource Usage	2	%

Figure 80 Memory Usage (XI50)

7.1.4. Debugging Capabilities

XI50 appliance can be debugged using:

- System log messages (Figure 81 depicts a sample system log)

time	category	level	tid	dir	client	msgid	message
Tue Apr 29 2008							
17:52:16	mgmt	notice	15631			0x8100000c	Saved current configuration to 'config:///tknap.cfg'
17:45:53	mgmt	notice	15631			0x8100000c	Saved current configuration to 'config:///tknap.cfg'
17:45:47	mgmt	notice	495			0x00350014	xmfirewall (XMarkOnion): Operational state up
17:45:47	mgmt	notice	495			0x00350016	xmfirewall (XMarkOnion): Service installed on port
17:45:47	mgmt	notice	495			0x00350015	xmfirewall (XMarkOnion): Operational state down
17:45:47	mgmt	warn	495			0x00340017	xmfirewall (XMarkOnion): Service removed from port

Figure 81 System Log (XI50)

- Probe - The probe enables observing all contexts of the used processing rule. Each context holds information about the current state of the processed XML document and about variables associated with the particular context (see Figure 82 and Figure 83).

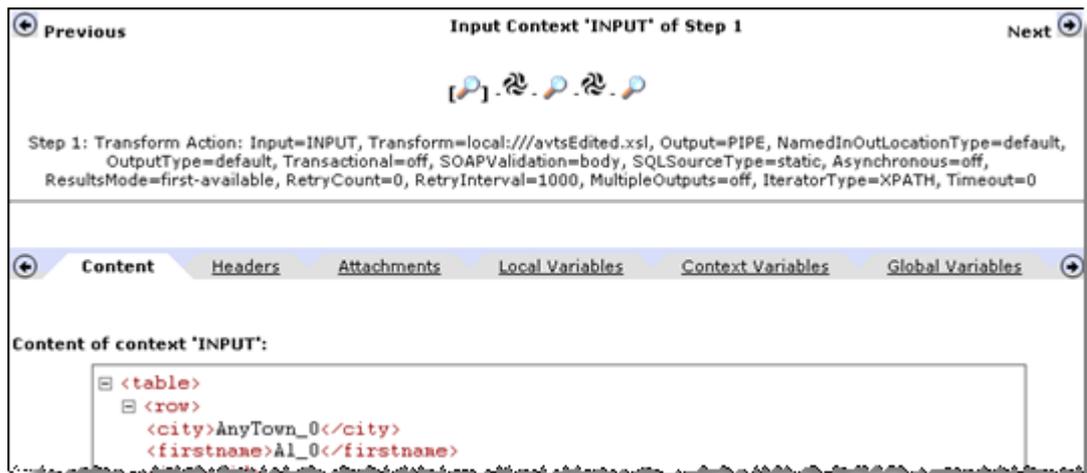


Figure 82 Probe - Selecting Context and Viewing XML Document

var://service/local-service-address	string	'9.138.237.68:50900'
var://service/max-action-depth	string	'128'
var://service/max-call-depth	string	'128'
var://service/multistep/contexts	node-set	(show nodeset)
var://service/multistep/input-context-name	string	'INPUT'
var://service/multistep/loop-count	string	(empty string)
var://service/multistep/loop-iterator	string	(empty string)

Figure 83 Probe - Context Variables

Obviously, during the testing, the log level is set to “fatal”, therefore, only the fatal errors are logged. Moreover, probe must be disabled, because it slows down the processing by tens of percents.

7.2. The SW Environment

Secondly, let us examine the settings of the SW environment. Chapter 7.2.1 holds configuration, Chapter 7.2.2 tuning possibilities, and Chapter 7.1.37.2.3 monitoring and debugging capabilities of this environment.

7.2.1. Configuration

Main Server with IBM WebSphere Application Server (School server):

- Intel Xeon CPU E5310@1.6GHz (4 Cores),
- 3.75GB RAM,
- HDD in RAID
- Windows Server 2003 R2, Standard Edition, SP2, 32-bit
- IBM WebSphere Application Server 6.1.0.2, Build cf20633.22 (18.8.2006)
- JRE 1.5.0 (build pwi32dev-20060511 (SR2)),
- IBM J9 VM (build 2.3, J2RE 1.5.0 IBM J9 2.3 Windows Server 2003 x86-32 j9vmwi3223-20060504 (JIT enabled)),

Auxiliary Server with Apache HTTP Server (R61-HTTP):

- Intel Core2 Duo CPU T7250@2.0GHz (2 Cores),
- 2 GB RAM,
- Windows Server 2003 R2, Standard Edition, SP2, 32-bit
- Apache HTTP Server 2.2.8.

Client (T40p-Client):

- Intel Pentium M @1.6 GHz (1 Core),
- 2 GB RAM,
- Microsoft Windows XP, SP2, with removed crippled TCP/IP stack (more than 10 TCP/IP connections can be opened at once)

Client (R61-Client):

- Intel Core2 Duo CPU T7250@2.0GHz (2 Cores),
- 2 GB RAM,
- Microsoft Windows XP, SP2 with removed crippled TCP/IP stack (more than 10 TCP/IP connections can be opened at once)

7.2.2. Tuning Possibilities

The tuning of the SW environment can have great impact on results of our tests. Especially the JVM settings are important.

IBM Java Virtual Machine (JVM) Tuning

This subchapter describes settings involving particular JVM parameters for running J2EE application on the Application Server.

To start with, the Java heap size has a direct impact on behaviour of a garbage collection and, consequently, on the performance of the running java application. The larger heap, the more memory is at our disposal, however, it takes also longer to accomplish a garbage collection. `Xms` switch specifies the initial java heap size, the `Xmx` switch the maximum heap size. For a production server, it is usually a good idea to specify the initial java heap size smaller than the maximum size, because the JVM heap can shrink and expand according to actual requirements. However, the changes in java heap size have measurable overhead, so for our case, where the optimal performance is required, the initial heap size and maximum heap size are set to the same value.

The default JVM heap size on a standard Java platform and 32-bit system is 64MB which is definitely not sufficient. A simple application executing XSL transformations with this default settings of the heap memory size shows that almost 30% of elapsed time is spent on garbage collections. When the maximum heap size of JVM is set to 1500MB, the time spent by garbage collecting is less than 0.001% of the summary elapsed time.

The default heap size of the Application Server is 768M, which should be increased as well. The maximum addressable memory on Windows Server 2003 32-bit is 4 GB (2^{32}). However, the operating system divides the total available physical memory into two parts - system space and user process space. Our main server has 3,75GB of physical memory, therefore, the theoretical maximum of memory consumed by an Application Server is about 1920MB. Hence, the amount of 1536MB of heap memory is a good candidate, because it is the double value in comparison with the default settings (768MB) and comprises 80% of the total available memory in the user space, which is a maximum recommended ratio of JVM heap memory size to the total available memory.

User space of an operational memory can be enlarged to the exclusion of the system space by using the switch `"/3GB` in the `boot.ini` file. After this modification, the maximum available memory for user process space increases to 2880MB. Therefore, the JVM heap size 2304MB is a suitable candidate, because it is a triple amount of the default heap memory of the

Application Server (768MB) and again comprises 80% of the total available memory in user process space (2880MB). However, this settings could be unstable due to restricted system space and are not considered in our testing.

JVM has a memory manager for automatic allocation and deallocation of memory on the heap, so that a programmer does not need to allocate a memory manually. When the JVM cannot allocate a memory for an object, the garbage collector (GC) is called to ensure the sweeping of the objects that are no longer needed. Each JVM vendor provides its own tuning parameters and garbage collector policies (the IBM JVM is not an exception).

`Xgcpolicy` switch specifies a policy used by a vendor specific garbage collector. The following four values can be used with IBM JVM:

- `optthroughput` (a default policy) - A garbage collector with this policy provides a high throughput, however the pauses during the garbage collection tend to be longer. Moreover, all application threads are stopped during a garbage collection.
- `optavgpause` - This policy collects garbage concurrently with the application execution, so that the garbage collection pauses are reduced. However, this policy has some impact on the overall throughput.
- `gencon` - This policy splits the heap into generation segments and uses two types of garbage collections - minor and major. A minor garbage collection takes place very often and operates over the young generation of objects (i.e. objects that were created since the last minor garbage collection). It uses the fact that many new objects live very shortly, so that they can be marked as garbage and swept very shortly after their creation during the next minor garbage collection. On the other hand, long-lived objects are promoted to the segment with old generation of objects. Major garbage collection maintains the old generation segment of the heap, when it is full. This policy is very useful for the stream processing of XML documents where the read and processed parts of the XML document can be very soon deallocated (in an ideal case till the next minor garbage collection)
- `subpool` - This policy is similar to the default one except that the heap is divided into multiple sections, so that scalability for object allocation is improved. It is designed primarily for systems with more than eight processors, thus this policy is not considered in our testing.

The impact of different settings of garbage collection policies together with both DOM and Stream parsing approaches is illustrated on the parsing of the batch of 100 test1.0.xml files using both parsing approaches. Figure 84 shows the total garbage collection runs according to various garbage collection policies and parsing approaches. Subsequently, Figure 85 depicts the time spend on one garbage collection. Finally, Figure 86 shows the percentage of time spent in the garbage collection of the overall duration of parsing and Figure 87 depicts “Requests per Second” measure for each garbage collection policy.

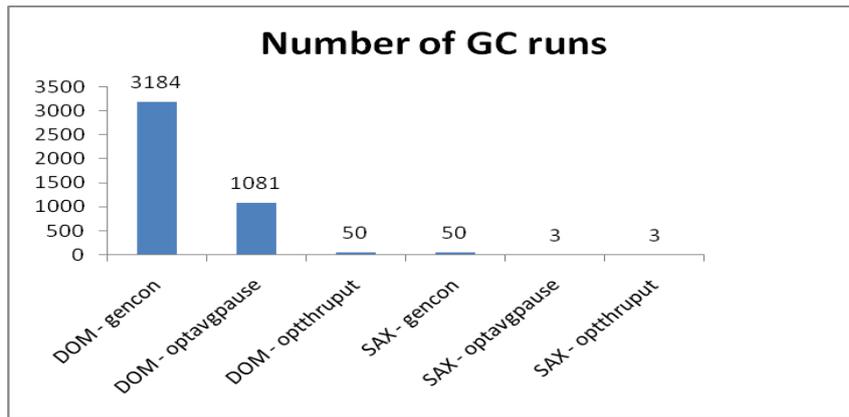


Figure 84 Number of GC Runs

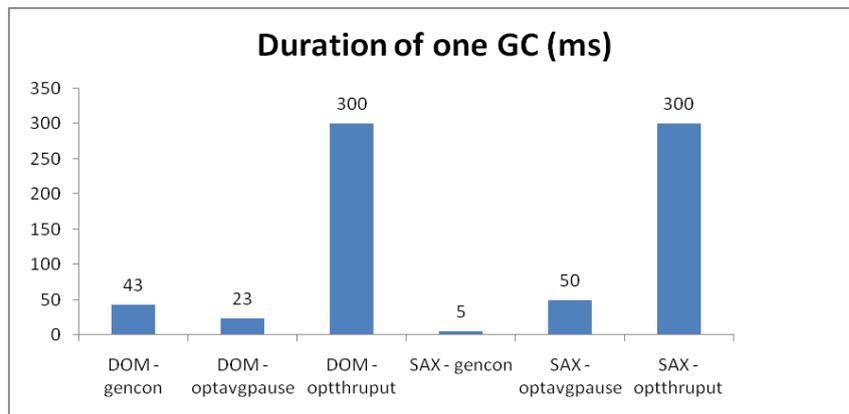


Figure 85 Duration of One GC Run

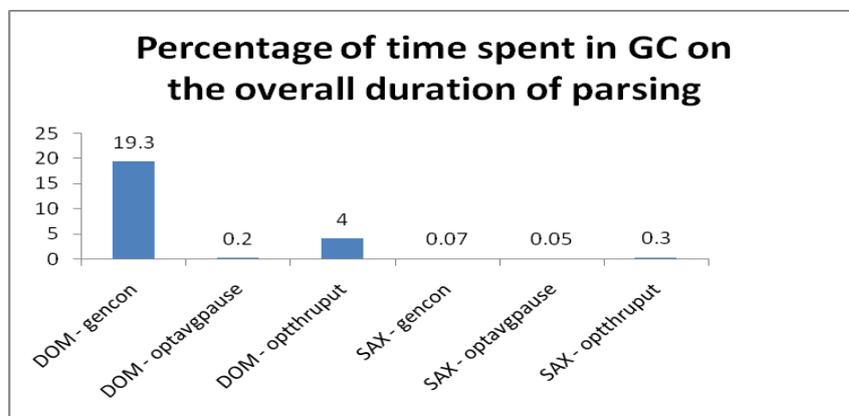


Figure 86 Percentage of Time Spent in GC

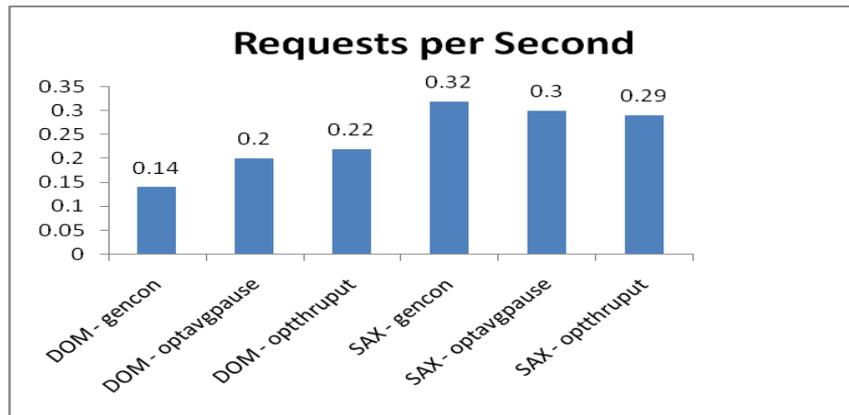


Figure 87 Requests per Second Measure

As we can see, more garbage collections do not inevitably mean worse results. The garbage collection policy should be used according to the character of the testing data. The `gencon` policy is very effective for Stream processing of XML documents, because many objects die until the next (minor) garbage collection. However, the DOM object tends to stay in the memory for a long time, thus the `gencon` policy for DOM processing gives very bad results, with more than 19% of total time spent in garbage collection. The policies `optavgpause` and `optthruput` tend to give similar results with reasonable time spent in garbage collection (less than 4% of the overall parsing time). The default policy `optthruput` tends to be optimal in DOM parsing and the worst in case of Stream processing, however the differences between the worst and the best are in Stream processing minimal. If we are looking for optimal policy for both parsing approaches, we definitely do not choose `gencon` policy. From the two resting policies, we choose `optthruput` policy, which gives better results in DOM processing used in major testing scenarios (All security actions must use DOM parsing).

Other JVM settings involve the following:

- `Xnoclassgc` switch - Standardly, the JVM unloads classes that are not used by any alive instances. The switch `-Xnoclassgc` disables this behaviour and, thus, increases the performance. However, this switch should be enabled only for special occasions, such as for our performance tests.
- `Xquickstart` switch - By default, JVM optimize the runtime performance, however, if this switch is present, the optimizing level is degraded in favour of a faster start of an Application server. This switch should be omitted, which gains slight performance boost.

Other Tuning Possibilities

- TCP/IP transport channel pool - The default value for the thread pool for incoming requests is 10-50 which should be expanded to 10-100.
- Servlet caching policy - Caching of servlets should be disabled, so that the result is always computed
- Keep-Alive interval - It specifies how often TCP repeats keep-alive transmission when no response is received (by default 2 hours). It helps keeping connection until the response is computed. We have tried to change this interval to 20-25s, so that the TCP/IP timeout does not occur in several test cases, however, we were not successful.
- “<http://apache.org/xml/properties/input-buffer-size>” is a property of Xerces parsing engine, which specifies a size of the input buffer in the XML Reader. Should be tuned

according to the size of the documents. Default value is 2KB, should not be more than 16KB. We do not experiment with this tuning possibility, however, it could slightly increase the speed of parsing.

There are far more settings, however, not all settings have measurable impact on the overall performance.

7.2.3. Monitoring and Debugging Capabilities

Firstly, all monitoring tools and the monitoring itself must be turned off during testing. This can be done in section “Monitoring and Tuning” and “Troubleshooting” of the web-based administration tool (see Figure 27). Specifically, Performance Monitoring Infrastructure (PMI) and Request Metrics (RM) tools should be disabled. PMI is a core technology for collecting underlying data of several other tools for viewing and analyzing performance. On the other hand, Request Metrics is a technology and tool for tracing individual transactions within the Application server.

Secondly, all in-depth logging tools in “Troubleshooting” section of the web-based administration tool (see Figure 27) must be disabled. This can be done by changing a log level from a default value (“info”) to an appropriate level (“fatal”), so that only fatal errors are logged. The logging could be switched off completely, however, this can overlook some fatal errors which can lead to misinterpretation of results.

7.3. Used Settings

Figure 88 depicts used settings of our testing environments during the testing.

Settings	Switches
1	See Chapter 7.1.2
A	-Xmx768M, -Xms768M, -Xnoclassgc
B	-Xmx 1536M, -Xms 1536M,-Xnoclassgc

Figure 88 Used Settings

In the HW environment, only Settings 1 are used (with possibility of enabling/disabling Streaming switch). In the SW environment, Settings A are the default settings suggested by the Application Server, with minimum heap size value set to maximum heap size value and with `noclassgc` switch enabled. Settings B have double amount of maximum and minimum heap memory in comparison with Settings A. Settings B are the default ones in the SW environment, if not specified otherwise.

8. TESTING

This section involves results of the testing of the “Flat” and “Onion” testing suite. As specified in Subsection 4.7, many measures are gathered with each test case run, however, this section analyzes only the major measure indicating the real-world performance - the throughput. Other measures (see Chapter 4.7.1) do not describe the performance of XML processing so aptly and/or the measures are dependent on the throughput measure, hence, these measures can be deducted from the throughput measure. All gathered measures can be viewed in reports appended to this thesis (see Subsection 0). If not specified otherwise, the throughput is measured in MB/s and the SW environment is using Settings B (see Subsection 7.3), which is shorten as SW_B.

Auxiliary files needed for the testing, such as XSL stylesheets, XSD files, and keys needed for securing actions, are cached. If these auxiliary files are loaded for every iteration of the same test case, it does not conform to a real-world situation where the repeatedly reused resources are typically cached and we want to measure metrics characterizing real-world performance. Moreover, in case of XSL stylesheets, the default XSLT engine of Java version 5 is XSLTC, which compiles (not interprets) XSL stylesheets. That kind of engine cannot be used in a reasonable way if the XSL stylesheet cannot be cached, and must be constantly reloaded and recompiled every time it is utilized.

As to the technique of gathering measures, the first few test cases run after starting the Application server or XI50 appliance last longer because of extensive loading of java classes and caching metadata about test cases. This is taken into account and all testing contain a few minutes of warm-up period containing loops of tests that we intend to measure. After this period, the system performance should be stabilized and the measures can be safely gathered. To add, all monitoring and debugging tools are disabled, all other processes which can influence the measured values are shut down. This is extremely important in the “Onion” testing suite where the test cases with the higher level of concurrency use fully all four cores of our School server and almost all or all resources on XI50 appliance.

In all subsections of Section 8, if we say that the throughput is in the environment A x times higher than in the environment B, we mean that the ratio of the average throughput in the environment A to average throughput in the environment B measured over the same set of test cases and possibly over other dimensions, such as tested engines, concurrency levels etc., is equal to x . If we talk about supported test cases, we mean test cases with testing data that can be successively run on the particular testing environment.

The network lines have a theoretical throughput of 1 Gb/s in both testing environments. Therefore, the utilization of the network is depicted in all subsections in Section 8 as a percentage of usage of 1 Gb bandwidth. In the “Onion” testing suite, the gathered values of the throughput measure can be in the HW environment even higher than 1Gb/s, because of the architecture of the testing framework - the throughput is measured together for two distinct sections of the network (see Subsection 4.4).

In all testing scenarios in the “Flat” testing suite except for the Parsing testing group, if the test cases with the XMark testing data ranging from Test0.0 to Test1.0 are supported, we say that all test cases are supported. In the Parsing testing group, even larger data are tested and evaluated using stream parsing approach. The restrictions on the supported test cases are in most cases caused by the “Out of Memory” exception when the heap size is not sufficient,

errors during processing testing or auxiliary data, TCP/IP timeouts, and/or by the limitations of AB and curl tools, which are both unable to upload very large files in the SW environment.

In the “Onion” testing suite, the test cases are chosen according to the results of the “Flat” testing suite and so that all of them are supported on all testing environments.

For each testing group, the lowest model of HW appliance (see Subsection 3.1) is specified.

8.1. The “Flat” Testing Suite

In the SW environment, the tests typically acquire fully one core of the School server. On the other hand, in the HW environment, the total workload of the general processing unit, accelerators, and other resources is typically between 20% and 30%. In scenarios running majority of the time on accelerators, the utilization of the general processor in XI50 appliance is less than 20%.

The memory of XI50 cannot be increased or decreased, the inner garbage collection policies cannot be changed. The approximate memory usage can be counted by multiplying the size of the input testing data by two (for input and output contexts of the processing rules) and, consequently, 1.5 should further multiply the result if the DOM processing technique is used. If the Stream processing approach is used, the memory requirements are insignificant even for very large files (tens of GBs)

In the SW environment, the memory and its management can be tuned, especially by setting the size of the accessible memory and choosing an appropriate garbage collection policy (see Chapter 7.2.2). The size of the heap memory depends on the selected settings of the SW environment.

The Parsing, Transforming, and Validating testing group can be successfully run on XA35, XS40, or XI50 appliance in the HW environment. The Securing testing group requires XS40 or XI50 appliance.

8.1.1. Parsing Testing Group

Parsing testing group covers DOM and Stream testing scenarios (see Section 5). Supported test cases differ according to the applied testing scenario and testing environment. The Stream testing scenario in the HW environment is utilizing all testing data, however, the Stream testing scenario in the SW environment and DOM testing scenarios in both testing scenarios do not use all testing data.

In the HW environment in DOM testing scenario, supported test cases are test cases containing any XMark file except of the file Test5.0 and larger. This restriction is enforced by the bounded memory of XI50 appliance with the respect to all aspects of the processing on XI50 appliance (see Subsection 4.5). In the SW environment, supported test cases are test cases ranging from Par_test0.0 to Par_test1.0 due to AB tool problem with uploading larger files. In addition, due to Out of Memory exception, DOM parsing with Settings A cannot process Par_test0.7 and Par_test1.0 test cases.

Firstly, let us examine the graph in Figure 89 depicting the throughput of the DOM and Stream testing scenarios in both environments (using Settings A and B in the SW environment) and test cases ranging from Par_test0.0 to Par_test1.0.

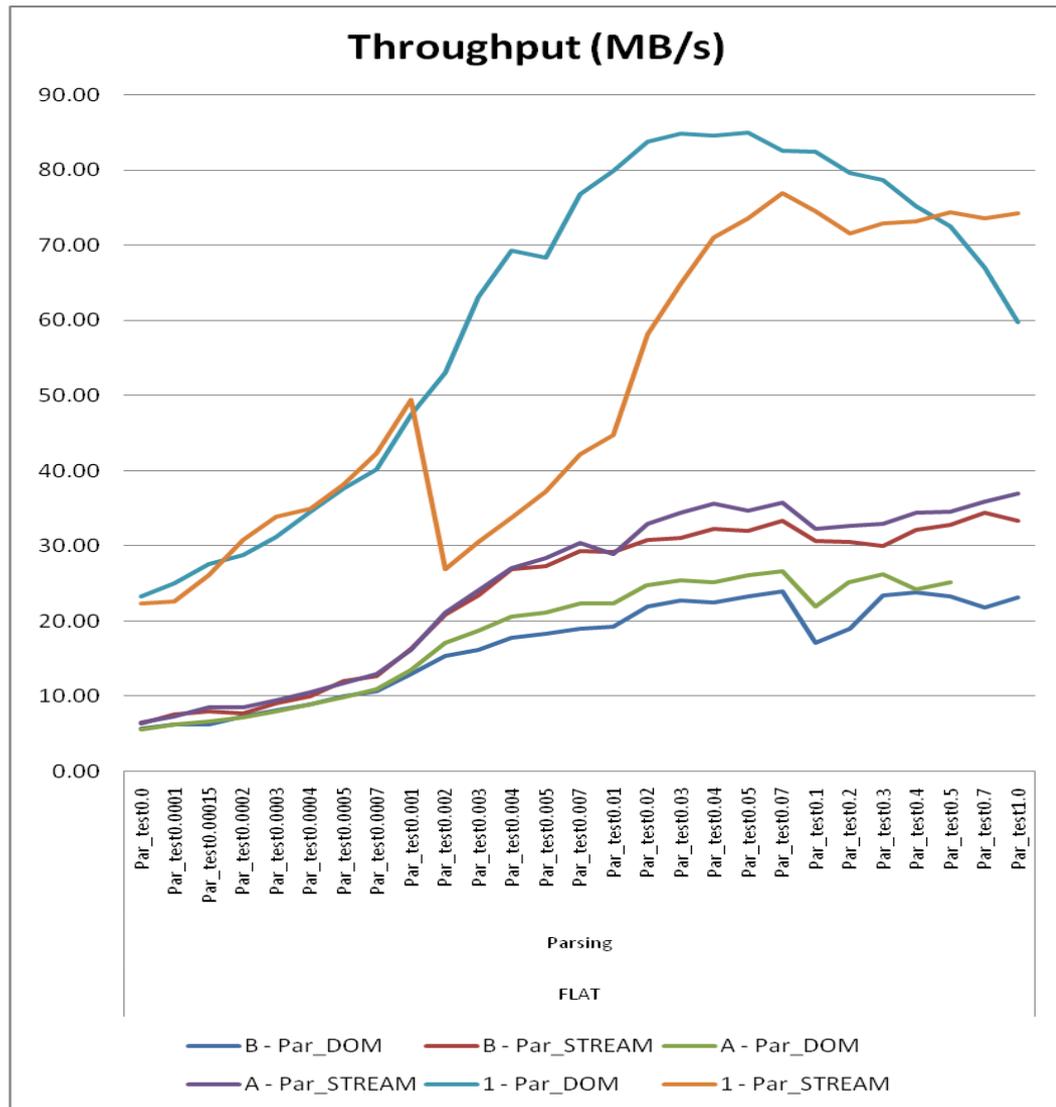


Figure 89 DOM and Stream Testing Scenarios in HW and SW

As we can see, the throughput of XI50 is higher in all test cases. To be more precise, the average throughput in case of DOM processing is in the HW environment approximately 3.76x higher than in SW-B environment (measured over all test cases of the DOM testing scenario ranging from Par_test0.0 to Par_test1.0). Figure 90 depicts the trend of the ratio of the throughput in the HW environment to throughput in the SW environment of DOM testing scenarios for the same test cases as in Figure 89.

Moreover, the throughput measured in the SW environment with Settings B and A does not differentiate much, which can lead us to the fact, that increasing the memory heap twice brings only a little gain (especially for smaller testing data) or even can bring worse results (especially for larger testing data) due to more time spent in the garbage collections. Therefore, the results gathered on XI50 with 4GBs of total memory and the SW environment with 768MB or 1536MB of heap size can be directly compared.

The throughput in the HW environment is highest for medium-sized XML documents (test cases ranging from Par_test0.01 to Par_test0.1) which is in coherence with the overhead of the TCP/IP stack transferring too small XML documents and with the network overhead of transferring large documents, where the TCP/IP stack tends to be ineffective. In the HW

environment, the DOM processing strategy is in general a better idea for testing data ranging from tens of kilobytes to tens of megabytes. Nevertheless, for the larger testing data, the Stream approach yields in better results (see test cases Par_test0.5, Par_test0.7, and Par_test1.0 in Figure 89), because of memory restrictions on XI50 appliance and subsequent garbage collections. The rapid decrease in the throughput in the HW environment in Stream testing scenario between Par_test0.001 and Par_test0.002 is a little bit mystery, however, the fact that DOM processing technique has better throughputs in most test cases (Par_test0.0 to Par_test0.4) is logical, due to quite large memory on XI50, accompanied by a significantly better memory management.

The throughput in the SW environment is not optimal for too small testing data as well, because of the network and application server overhead. Nevertheless, the throughput is not falling down for large files as in the HW environment, which is possibly due to not so effective parsing technique of the general CPU on the Application server. Therefore, the curve will possibly start falling down only for even more larger files which are out of the scope of our supported test cases. In the SW environment, The DOM parsing is definitely more resource demanding than the Stream processing which has 1.38x higher average throughput than the DOM parsing approach (measured over all test cases ranging from Par_test0.0 to Par_test1.0 and using Settings B in both cases).

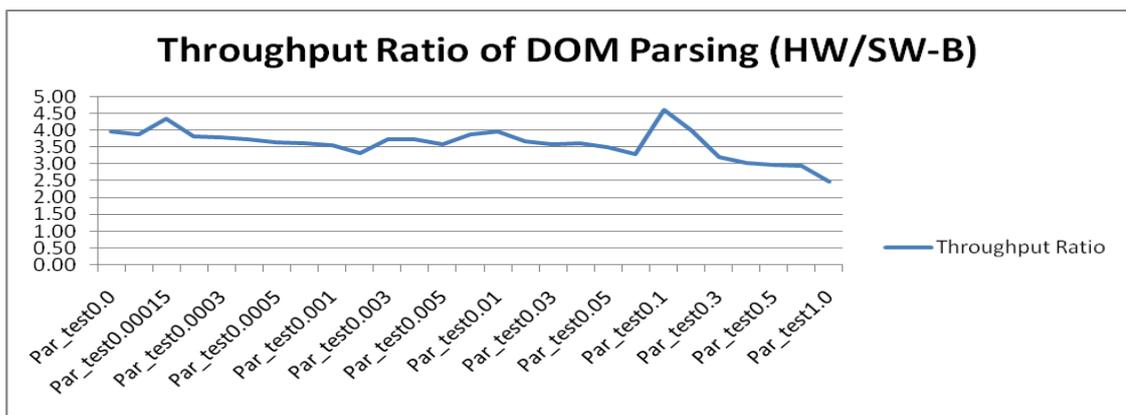


Figure 90 Throughput Ratio of DOM Parsing

8.1.2. Validating Testing Group

Validating testing group involves four testing scenarios containing DTD and XML Schema validations (see Section 5). Supported test cases differ according to the testing scenario and the testing environment. In the SW environment, all testing scenarios are supported. In the HW environment, Val_DTD_BASE testing scenario is not supported at all, testing scenario Val_XSD_REDUCED supports all test cases, and testing scenarios Val_XSD_BASE and Val_XSD_EXTENDED are successfully run for all test cases except for test case Val_test1.0.

Figure 91 depicts the throughput of all testing scenarios in the Validating testing group in both environments and for test cases ranging from Val_test0.0 to Val_test1.0.

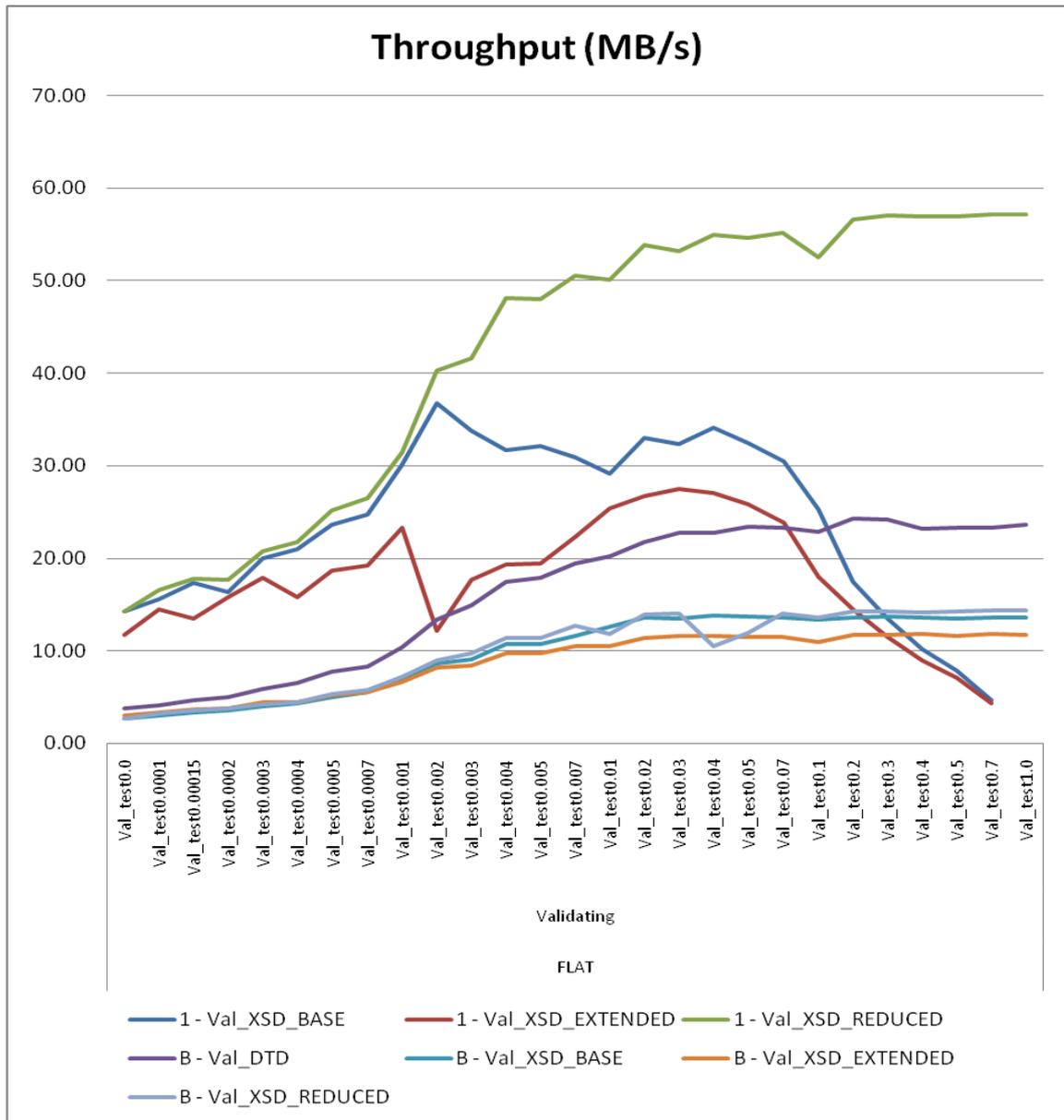


Figure 91 Validating Testing Group in HW and SW

The throughput on XI50 appliance is highly dependent on the measured scenario. For smaller test cases (Val_test0.0 to Val_test0.002) the Val_XSD_BASE and Val_XSD_REDUCED have similar throughputs, however, with the growing size of the testing data, the throughput is beginning to differ more and more. This shows that introducing ID and IDREF attributes in XSD files is very resource demanding for larger files and influences the final throughput significantly. In our testing scenario even more than adding some pattern matching on regular expressions (compare differences between Val_XSD_BASE - Val_XSD_EXTENDED and Val_XSD_BASE - Val_XSD_REDUCED testing scenarios). The throughput of Val_XSD_BASE and Val_XSD_EXTENDED testing scenarios for test cases Val_test0.4 to Val_test0.7 is even worse than in the SW environment for the same test cases. This affirms the latest trend of XI50 appliance to be rather message-oriented than batch-oriented appliance with strong orientation on validating SOAP messages which are typically smaller than tens of megabytes.

On the other hand, in the SW environment, DTD validation is significantly faster (with almost double throughput than other validating testing scenarios), Val_XSD_BASE and Val_XSD_REDUCED testing scenarios have similar throughputs, only the throughput of the testing scenario Val_XSD_EXTENDED is little bit lower. Therefore, contrary to the results in the HW testing environment, the difference between testing scenarios Val_XSD_BASE and Val_XSD_EXTENDED is greater than between Val_XSD_BASE and Val_XSD_REDUCED testing scenarios. Moreover, the complexity of XSD validations has significantly smaller influence on the resulting throughput than in the HW environment. This relates with results in Figure 89 showing lower efficiency in parsing XML documents.

The comparison of throughputs of the Parsing and Validating testing groups is covered in Subsection 8.1.5

8.1.3. Transforming Testing Group

Transforming testing group contains four testing scenarios governing XSLT transformations (see Section 5). All test cases are executed properly except for the test case “Tra_number” on SAXON-B engine in the SW environment, which fails due to fatal error during parsing XSL stylesheet `number.xsl`.

Figure 92 and Figure 94 reveal the measured throughput of Tra_XSLTMark and Tra_XSLTMark_XL testing scenarios in both environments and with all test cases defined in Section 5.

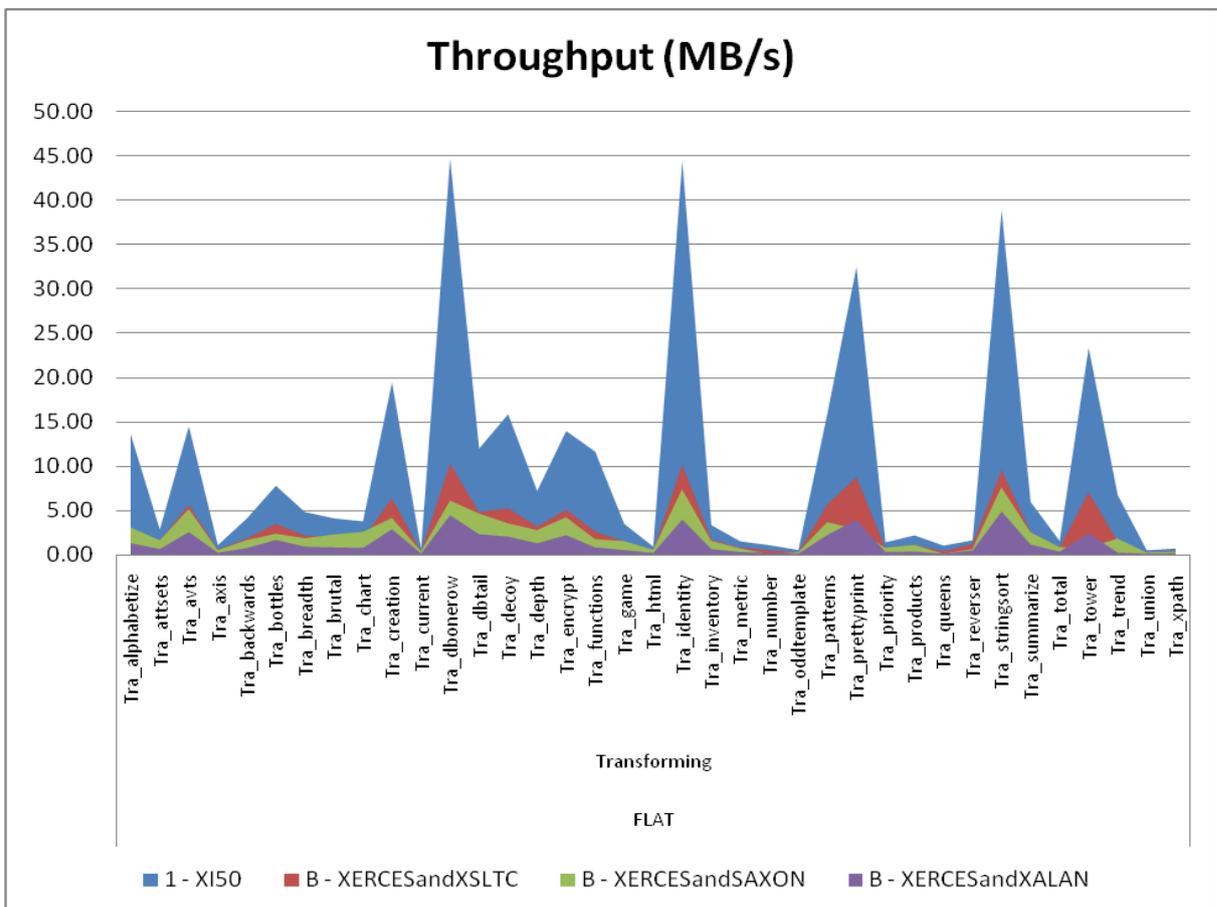


Figure 92 XSLTMark in HW and SW

As we can see, XI50 appliance has better throughputs in all test cases, the average throughput in the HW environment is 4.64x higher than in the SW environment (measured over all test cases of XSLTMark over all engines in the HW and SW environments).

The differences are noticeable especially in test cases containing complex and/or numerous XPath queries, because XI50 appliance owns a memory addressable by XPath queries. For example, the throughput of test case Tra_functions has in the HW environment 8.32x higher average throughput than in the SW environment (measured over all test case runs of Tra_functions test case from the Tra_XSLTMark testing scenario and over all engines in the HW and SW environments). With the increased size of the input testing data, the differences between environments are even higher - see Figure 94 containing throughputs of the Tra_XSLTMark_XL testing scenario.

On the field of SW XSLT engines, XSLTC has the highest throughput in most cases, especially in cases of complex XSL stylesheets, such as in Tra_tower test case, where the compiled XSL stylesheet is much more effective than the interpreted one. SAXON-B beats XSLTC in some test cases involving simple XSL stylesheets, because XSLTC shares many libraries with XALAN engine, which is the worst XSLT engine in most test cases possibly due to ineffective memory management. It is especially noticeable in Tra_identity test case that actually performs only identity transformation. Figure 93 shows memory allocation behaviour of both engines during execution of all test cases in Tra_XSLTMark testing scenario on XALAN and SAXON-B engine. The horizontal axis shows the time and the vertical axis megabytes of allocated memory. The peak values are ranging from 0 to 500MBs of allocated memory.

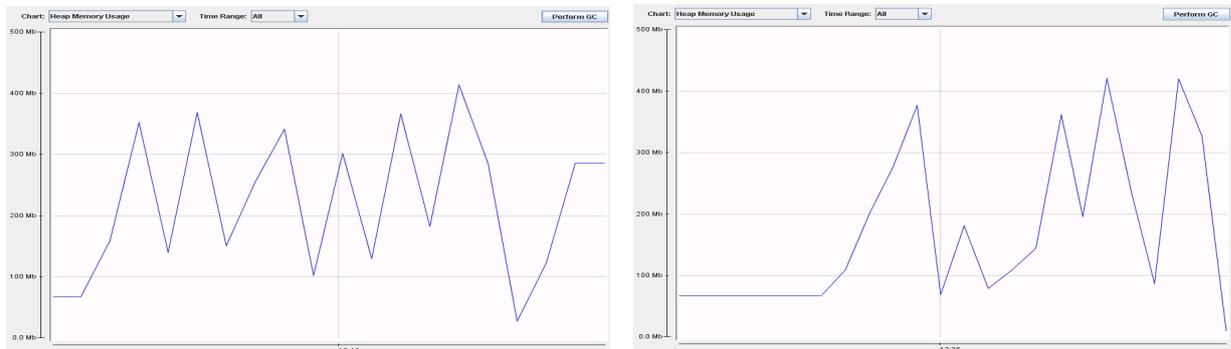


Figure 93 XALAN (on the Left) and SAXON-B (on the Right) Heap Usage

If we take a closer look at both graphs, we can see that the XALAN engine is definitely allocating more memory on the heap (about 200-300MB more than SAXON-B). Such phenomenon indicates larger/more objects created during execution of tests.

XALAN has better result than SAXON-B in some test cases requiring loading of the whole input document to memory (such as Tra_encrypt), where the memory overhead cannot be bypassed.

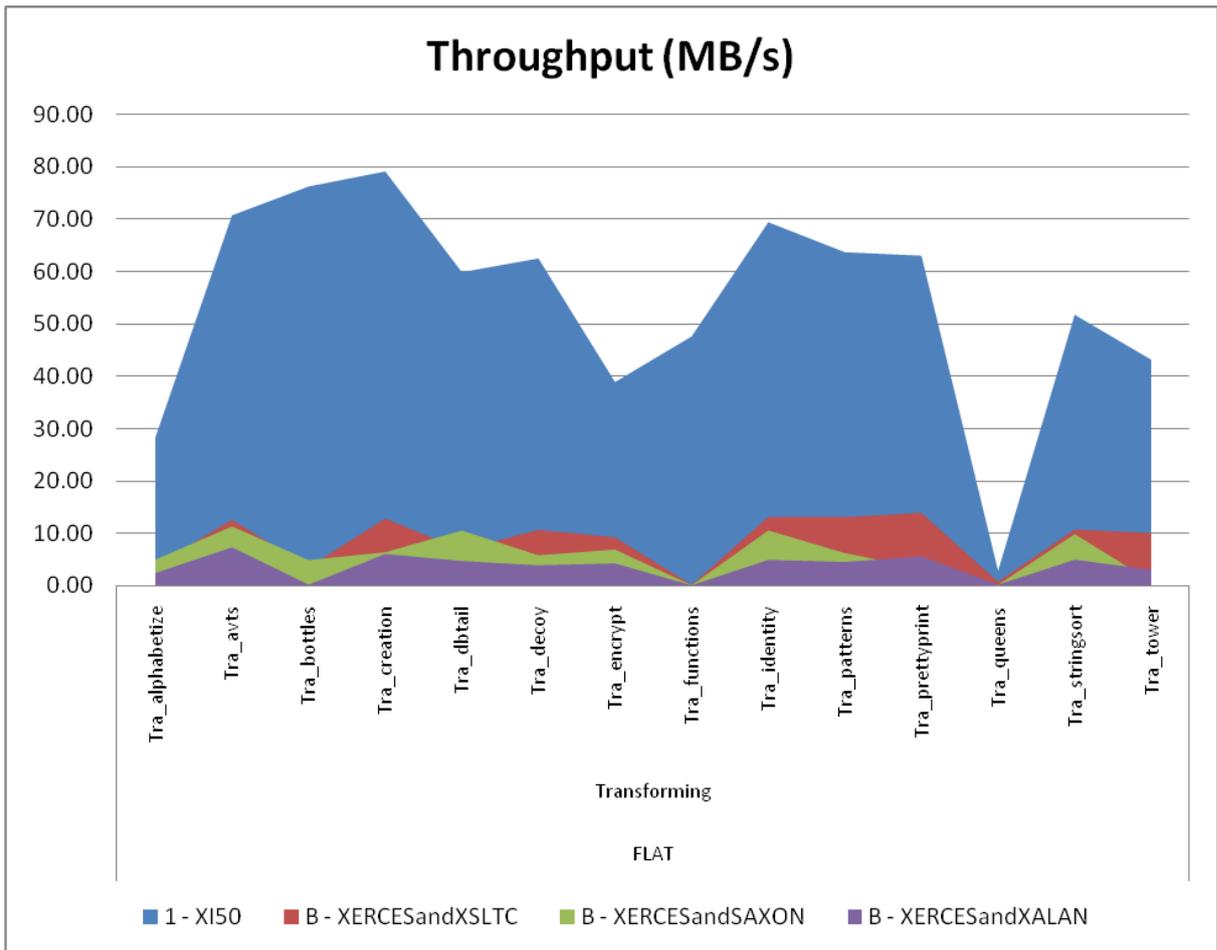


Figure 94 Comparison of Throughputs of XSLTMark_XL

Figure 94 shows test cases of Tra_XSLTMark_XL testing scenario, where the differences in throughputs in the HW and SW environments are even more obvious, because of larger testing data and, hence, lower network overhead. For example, the test case Tra_functions has 1535x higher average throughput in the HW environment than in the SW environment (measured over all test case runs of test case Tra_functions of XSLTMark_XL testing scenario and over all engines in the HW and SW environments).

Figure 95 contains summary comparison of throughputs in the HW environment of test case supported by all testing scenarios in Transforming testing group. For each test case, its throughput is typically highest for Tra_XSLTMark_L or Tra_XSLTMark_XL testing scenarios.

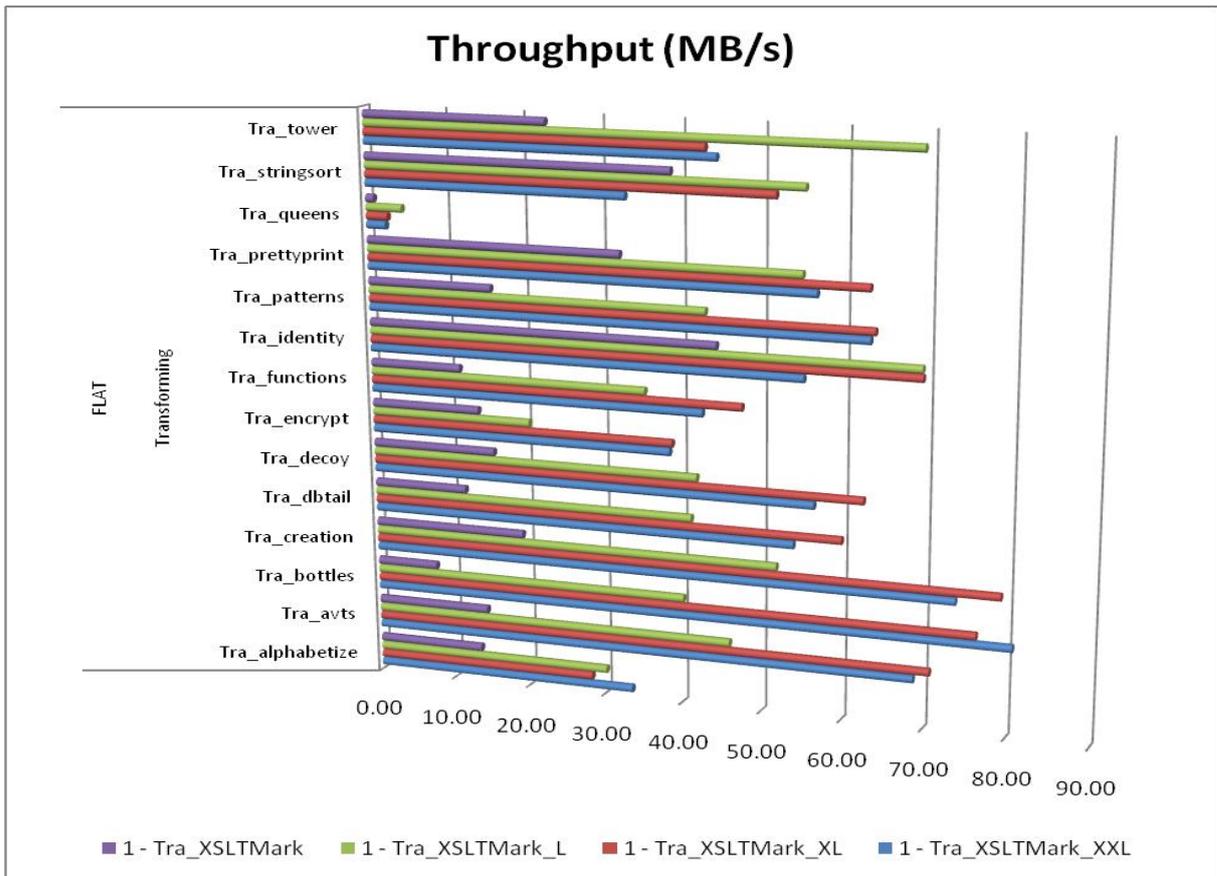


Figure 95 Comparison of Throughputs of Transforming Testing Group in HW

8.1.4. Securing Testing Group

Securing testing group involves testing scenarios containing encrypting and decrypting using different cryptographic algorithms (see Section 5). Firstly, let us look at encrypting and decrypting testing scenarios followed by signing and verifying testing scenarios.

Encrypting and Decrypting Testing Scenarios

In the SW environment, Sec_ENC_ENC_RSA2_* and Sec_ENC_DEC_RSA2_* testing scenarios support all test cases except for Sec_test0.7 and Sec_test1.0 due to Out of Memory exception (token '*' denotes both variants of encrypting and decrypting using AES or 3DES algorithms). In the HW environment, all testing scenarios support all test cases, except for the testing scenarios Sec_ENC_DEC_RSA2_AES256 and Sec_ENC_DEC_RSA2_3DES which do not support test cases ranging from Sec_test0.4 to Sec_test1.0 due to TCP/IP timeout.

Figure 96 depicts the measured throughput of selected encrypting testing scenarios in both environments and for test cases ranging from Sec_test0.0 to Sec_test0.5.

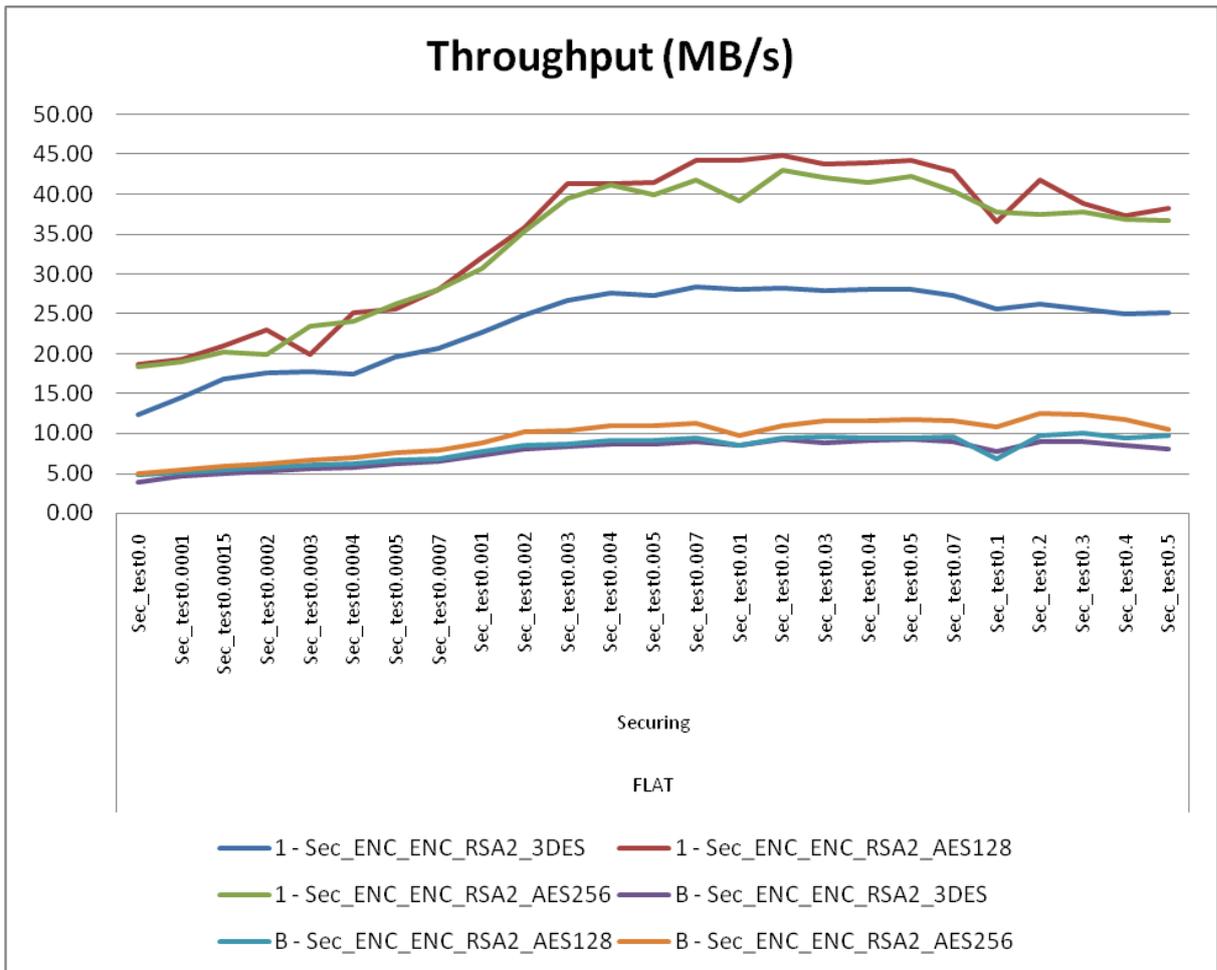


Figure 96 Encrypting in HW and SW

In the SW environment, the choice of a symmetric cipher used during the encryption of the input testing data is rather insignificant. On the other hand, XI50 has almost double throughput when using AES algorithm instead of 3DES for encryption, regardless the size of the key used (128 or 256 bits).

Due to security accelerators on XI50 appliance, the throughput in the HW environment is almost triple in average in comparison with the SW environment. The trend of all values belonging to one testing scenario copies the trend in Figure 89, because encrypting requires to have a DOM tree built in a memory, hence, involves DOM processing.

Using RSA (utilizing 1024-bits long key) instead of RSA2 (RSA with 2048-bits long key) gives similar results, which is logical, because an asymmetric key is used only for encrypting the symmetric key used to actual encryption of the XML document (see 2.1.4) and therefore the results with RSA used instead of RSA2 are not included in Figure 96. In the “Onion” testing suite, RSA2 is preferred due to higher security. As to the symmetric algorithms used, we prefer AES algorithm with 256-bits long key, which has better throughputs in both testing environments and is adopted as an encryption standard by the U.S. government.

Sec_ENC_ENC_PART_RSA2_AES256 testing scenario is measured only on XI50 and gives worse results than Sec_ENC_ENC_RSA2_AES256. Therefore, it is better and preferred to encrypt the whole XML document instead of, for example, all occurrences of an element

“Age” within the collection of elements “Person”. In the SW environment, this testing scenario could not be measured, because of lack of documentation in IBM encryption Java library.

As to decrypting testing scenarios, both have in SW as well as in the HW environment worse throughputs than the appropriate encrypting scenarios. To be more precise, In the HW environment, encryption using RSA2 and AES256 algorithms has 1.51x higher average throughput and encryption utilizing RSA2 and 3DES algorithms has 1.37x higher average throughput than decryption using the same algorithms and measured over all supported test cases. In the SW environment, encryption using RSA2 and AES256 algorithms has 2.85x higher average throughput and encryption with RSA2 and 3DES algorithms has 2.16x higher average throughput than decryption using the same algorithm and measured over all test cases.

Signing and Verifying Testing Scenarios

In the SW environment, Sec_SIG_SIG_RSA_SHA1 and Sec_SIG_SIG_RSA2_SHA1 testing scenarios support all test cases except for Sec_test0.5, Sec_test0.7, and Sec_test1.0, testing scenario Sec_SIG_SIG_DSA_SHA1 supports all test cases except for Sec_test0.7 and Sec_test1.0, and, finally, Sec_SIG_VER_DSA_SHA1 supports all test cases. Unsupported test cases are due to Out of Memory exception. In the HW environment, all testing scenarios support all test cases, except for the testing scenario Sec_SIG_VER_DSA_SHA1 that does not support test cases ranging from Sec_test0.2 to Sec_test1.0 due to TCP/IP timeout.

Figure 97 depicts the throughput of selected signing testing scenarios in both environments and for test cases ranging from Sec_test0.0 to Sec_test1.0.

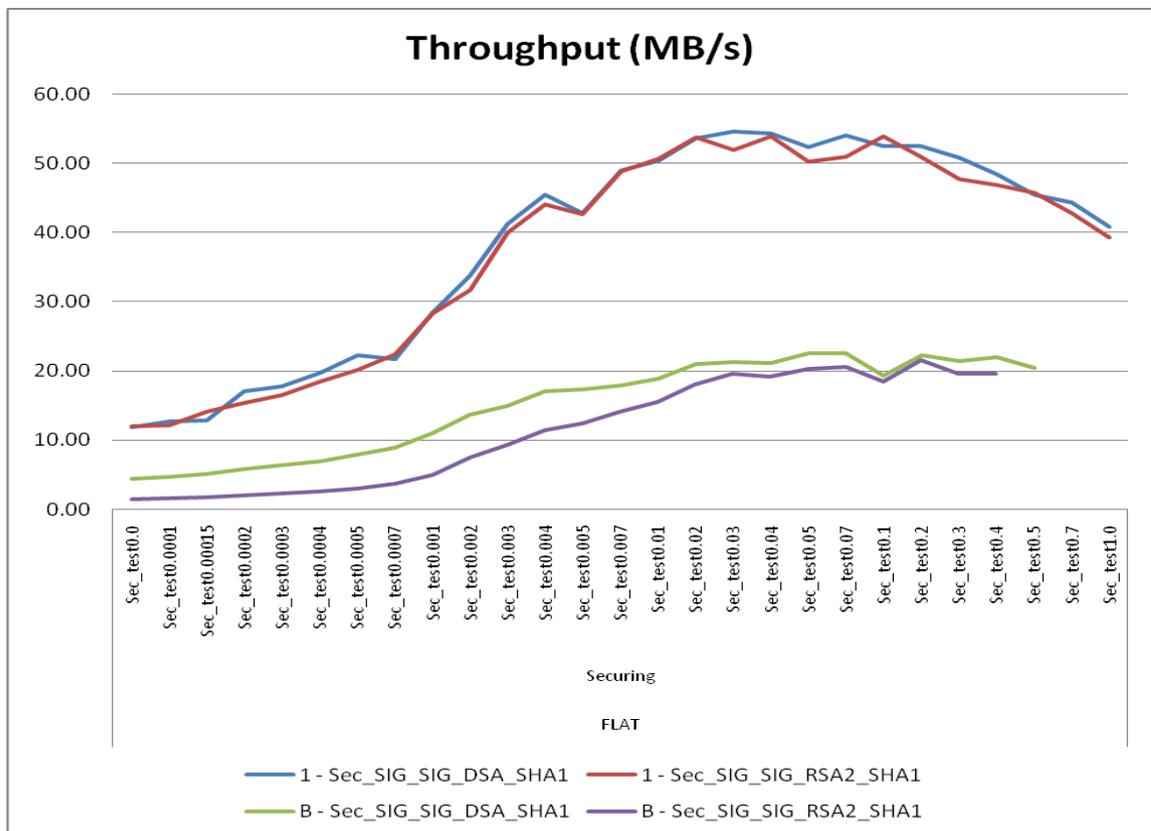


Figure 97 Signing in HW and SW

In both environments, the choice of the asymmetric algorithm used for signing the digest of the XML document is rather unimportant. This is logical, because asymmetric algorithm is used only for signing the message digest, not the whole message. However, the combination of DSA and SHA1 algorithms is preferred, because DSA and SHA1 are both required by XML Signature standard and, what is more, DSA is United States Federal Government standard for digital signatures.

The average throughput of signing action is in the HW environment 3.54x higher than in the SW environment (measured over all test cases ranging from Sec_test0.0 to Sec_test0.5 and over both testing scenarios Sec_SIG_SIG_RSA2_SHA1 and Sec_SIG_SIG_DSA_SHA1). The trend of all values in Figure 97 belonging to the same testing scenario copies the trend of DOM parsing testing scenario in Figure 89, because signing (as well as encrypting) requires a DOM tree in a memory and, therefore, DOM processing precedes the signing action.

As to verifying testing scenarios, in the HW environment, Sec_SIG_SIG_DSA_SHA1 has 14.28x higher average throughput than verifying using the same algorithms (measured over all test cases ranging from Sec_test0.0 to Sec_test0.1). In the SW environment, Sec_SIG_SIG_DSA_SHA1 has 1.38x higher average throughput than verifying using the same algorithms (measured over all test cases ranging from Sec_test0.0 to Sec_test0.5).

Comparison of Encrypting and Signing

Figure 98 is comparing throughputs of signing using DSA and SHA1 algorithms and encrypting using RSA with 2048-bits long key and AES with 256-bits long key in both testing environments.

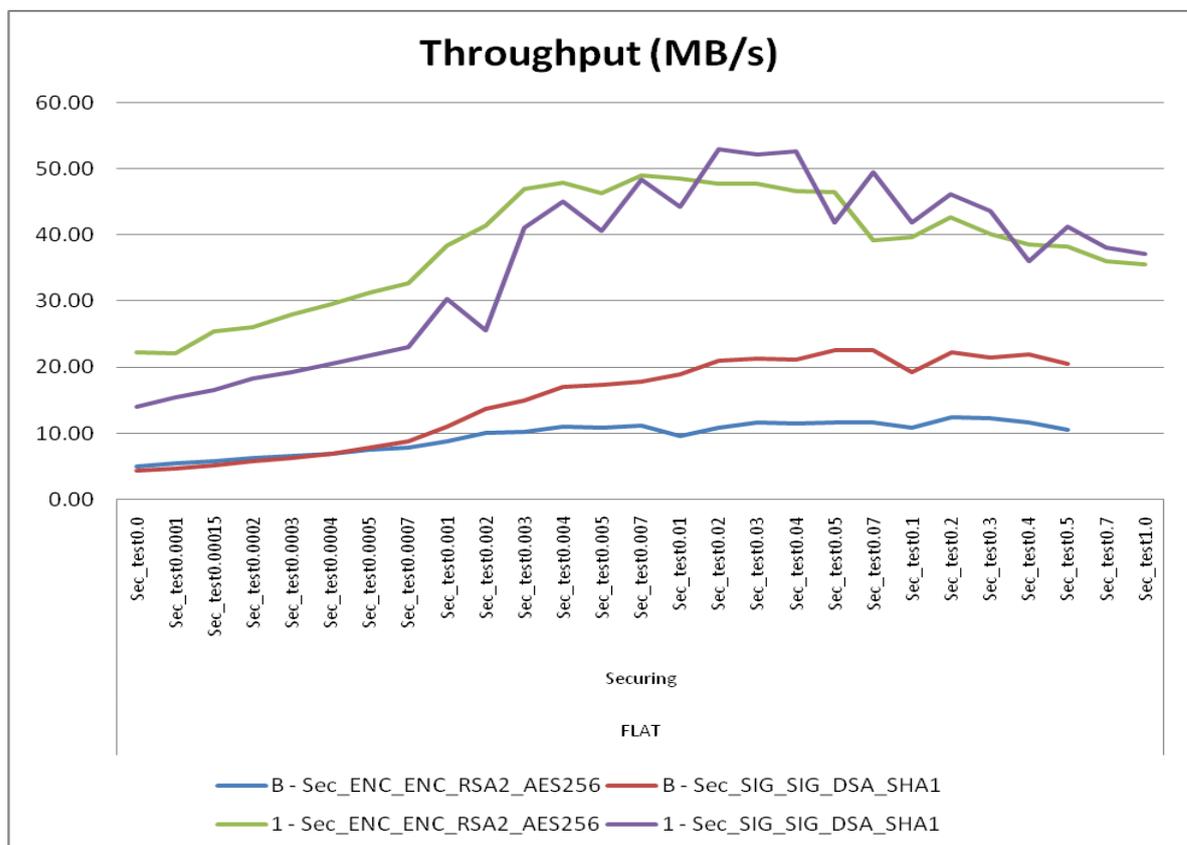


Figure 98 Comparison of Signing and Encrypting in HW and SW

In the HW environment, signing and encrypting (for test cases from Sec_test0.004 to Sec_test1.0) has similar throughputs. In the SW environment, the throughput of signing action is almost double for test cases from Sec_test0.01 to Sec_test1.0.

8.1.5. Cross-scenario Comparison

Figure 99 and Figure 100 compare throughputs of selected testing scenarios from Parsing, Validating, and Securing testing group separately in the HW and SW environment.

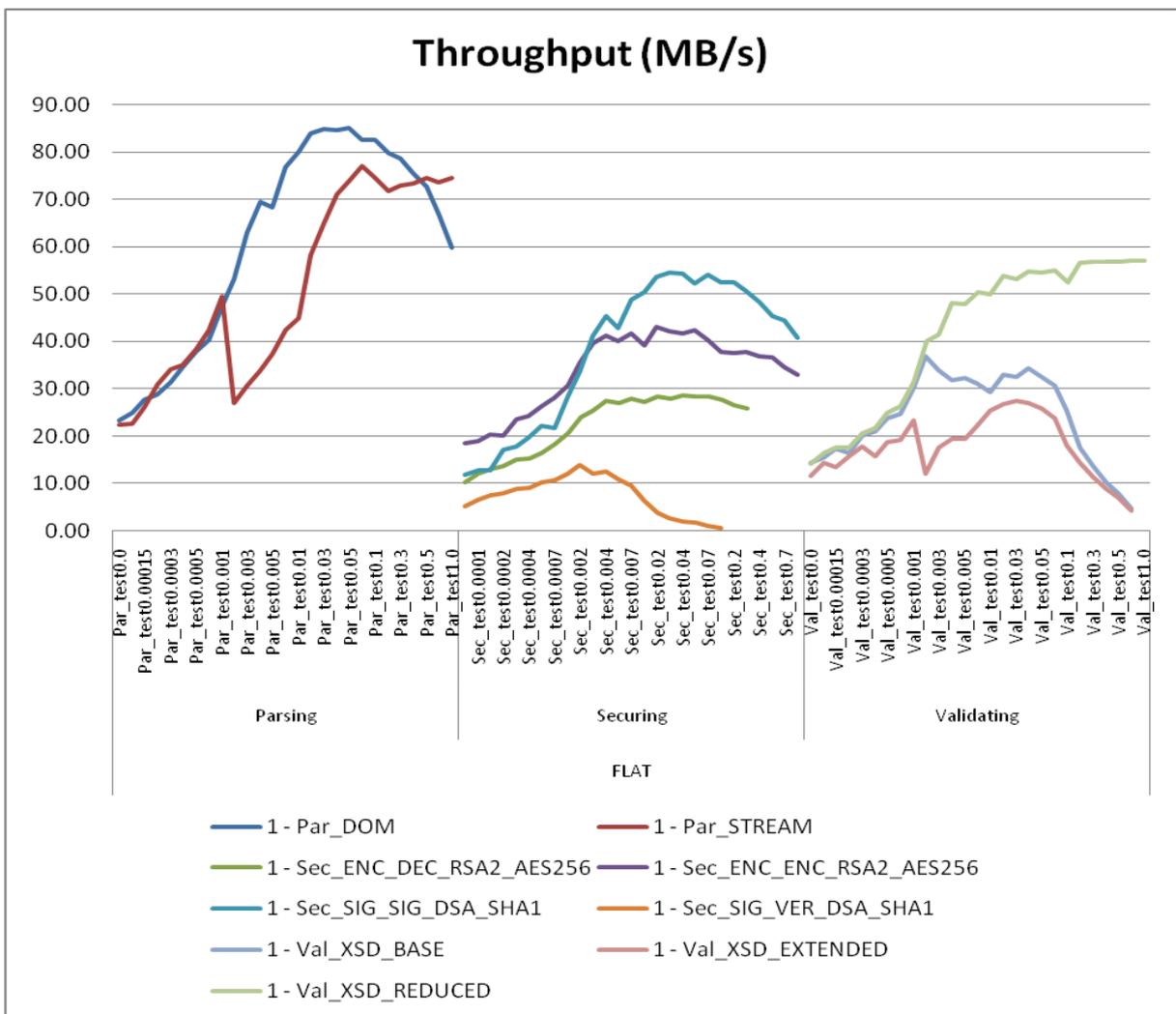


Figure 99 Summary Comparison of Throughputs in the HW Environment

As we can see, in the HW environment, Parsing testing group has highest maximum throughputs, followed successively by the testing scenarios Sec_SIG_SIG_DSA_SHA1, Sec_ENC_ENC_RSA2_AES256, and Val_XSD_REDUCED. Moreover, the shape of the curves of the Securing testing group is similar to the one of the Par_DOM testing scenario and the shape of the curves of the Validating testing group is like the one in Par_Stream testing scenario. This is because DOM parsing is preceding all Securing actions and, on the other hand, Validating actions are processed in a streamed way.

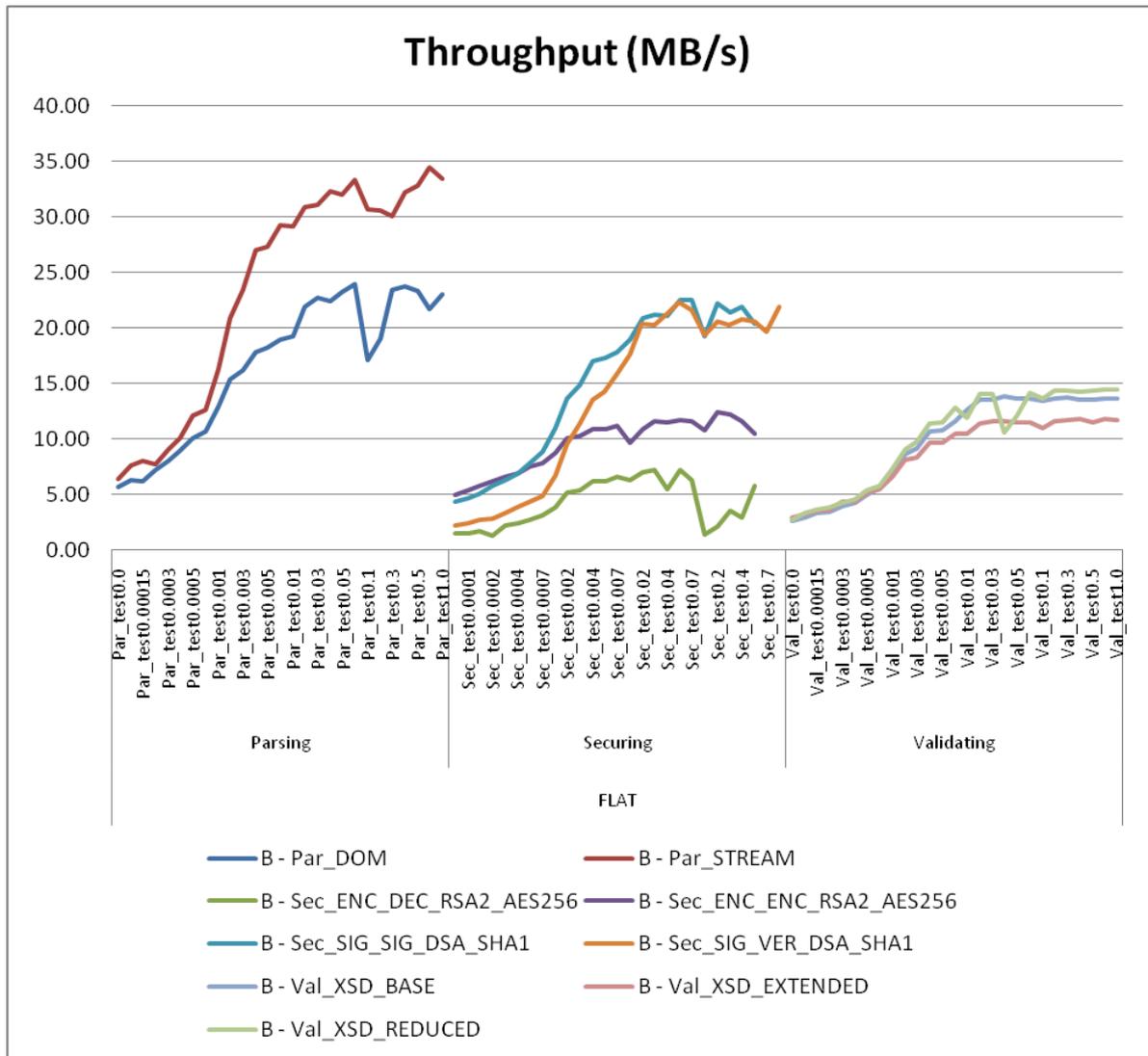


Figure 100 Summary Comparison of Throughputs in the SW Environment

In the SW environment, the similarities in the shapes of the curves are not as visible as in Figure 99 due to smaller differences in throughputs of different sizes of testing data of one testing scenario. However, we can see that signing and encrypting has higher throughputs than any testing scenario from the Validating testing group containing XML Schema validation.

Network Utilization

Finally, let us have look at the percentage utilization of 1Gb network by various chosen testing scenarios. For each testing scenario, except for Transforming testing group, the maximum average test case throughput is counted. In case of Tra_XSLTMark testing scenario, the average throughput of five test cases with the lowest/highest average throughputs is counted and marked as Tra_XSLTMark_Low/Tra_XSLTMark_High in the table in Figure 101. Finally, for the Tra_XSLTMark_XL testing scenario, the average throughput of three test cases with the lowest/highest average throughputs is counted and marked as Tra_XSLTMark_XL_Low/ Tra_XSLTMark_XL_High in Figure 101.

Figure 101 shows the collected results. Network usage less than 10% is emphasized by red colour and network utilization over 50% is marked by green colour. Figure 102/Figure 103

shows the percentage throughputs in comparison with 1Gb network in both environments of all chosen testing scenarios sorted according to reached results in HW/SW environment.

Testing Scenario	SW - % of 1Gb	HW-% of 1Gb
Par_DOM	18.7 (Par_test0.07)	66.4 (Par_test0.05)
Par_STREAM	26.9 (Par_test0.7)	60 (Par_test0.07)
Val_DTD_BASE	18.9 (Val_test0.2)	N/A
Val_XSD_BASE	10.8 (Val_test0.04)	28.7 (Val_test0.002)
Val_XSD_REDUCED	11.3 (Val_test0.7)	44.7 (Val_test0.7)
Val_XSD_EXTENDED	9.2 (Val_test0.7)	21.5 (Val_test0.03)
Tra_XSLTMark_High	4.3	28.7
Tra_XSLTMark_Low	0.1	0.6
Tra_XSLTMark_XL_High	6.5	58.9
Tra_XSLTMark_XL_Low	0.5	18.2
Sec_ENC_ENC_RSA2_AES256	9.7 (Sec_test0.2)	33.6 (Sec_test0.02)
Sec_ENC_DEC_RSA2_AES256	5.6 (Sec_test0.03)	22.3 (Sec_test0.04)
Sec_SIG_SIG_DSA_SHA1	17.6 (Sec_test0.05)	42.7 (Sec_test0.03)
Sec_SIG_VER_DSA_SHA1	17.4 (Sec_test0.05)	10.8 (Sec_test0.002)

Figure 101 Network Utilization of the “Flat” Testing Suite with Optimal Test Cases

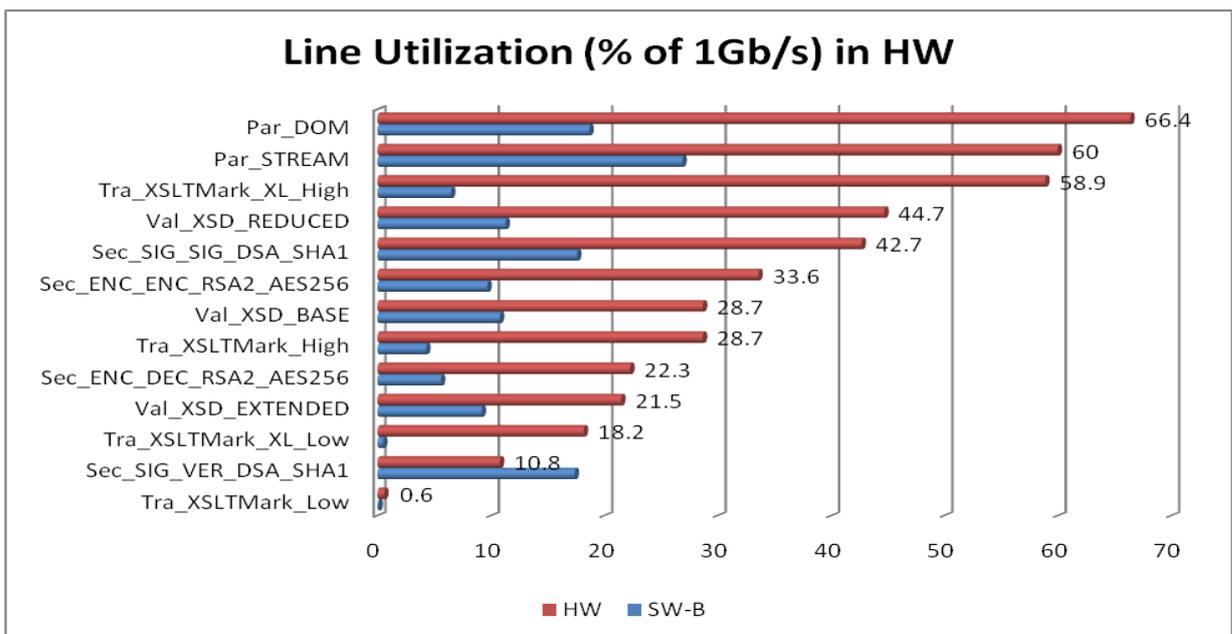


Figure 102 Network Utilization of the “Flat” Testing Suite in HW

In the HW environment, the three highest throughputs belong to both Parsing testing scenarios and to the average throughput of three test cases with the highest throughput from the testing scenario Tra_XSLTMark_XL containing easy XSL stylesheets and relatively large testing data (ranging from hundreds of kilobytes to megabytes). All these three testing scenarios have its maximum peak utilization over 50% without counting any concurrency.

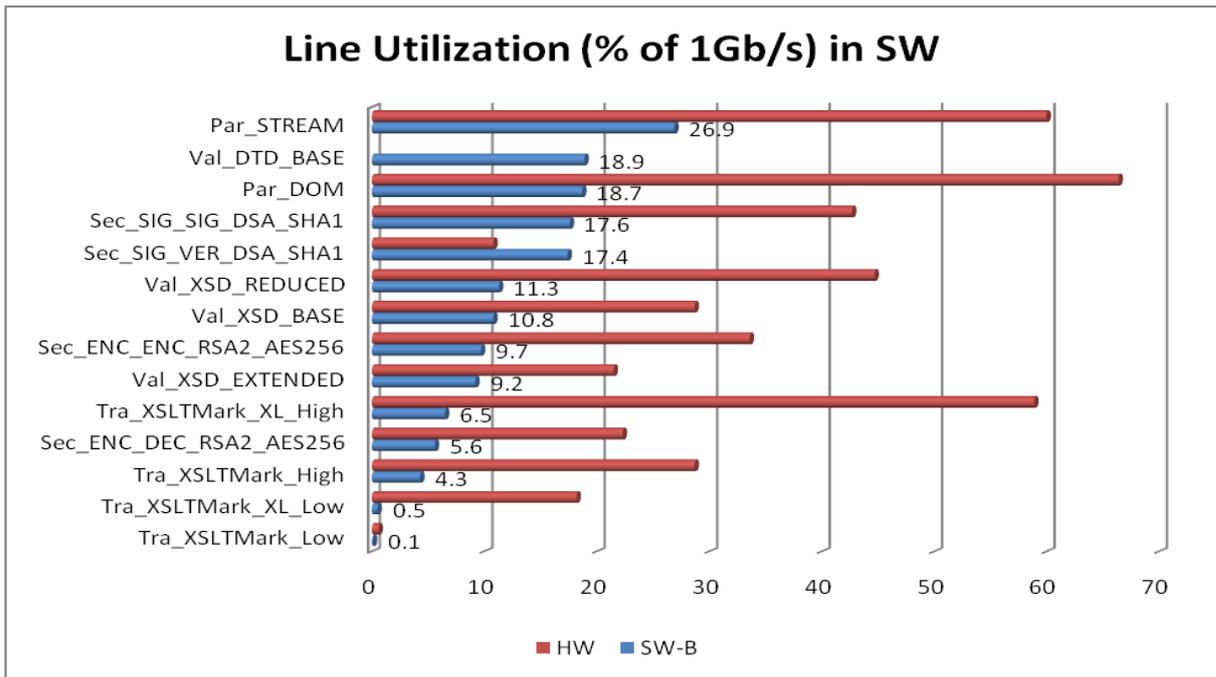


Figure 103 Network Utilization of the “Flat” Testing Suite in SW

On the other hand, in software environment, the first three ranks of the scale are occupied by Parsing testing scenarios and Val_DTD_BASE testing scenario. If we compare the maximum throughput of Val_DTD_BASE and Val_XSD_BASE testing scenarios, we can see one of the reasons, why DTD validation is still popular. Besides, we can see, how many testing scenarios in the SW environment have the maximum throughput less than 10% of the line capacity.

The highest contrast is in the testing scenario Tra_XSLTMark_XL, where the average throughput of the three slowest test cases (Tra_XSLTMark_XL_Low) is in the HW environment still three times higher than the average throughput of the three fastest test cases (Tra_XSLTMark_XL_High) in the SW environment. To sum up, the greatest weakness in the SW environment is the throughput of tough XSL transformations of testing data of all sizes, the utilization is under 0.5%, which introduces serious bottleneck in processing XML documents.

8.2. The “Onion” Testing Suite

In both testing environments, the usage of general CPU is ranging from a few percents to 100 percents. However, in the HW environment, the maximum load is typically reached with more concurrent requests and with growing complexity of testing scenarios, the general CPU usage tends to be lower due to help of accelerators.

As to memory requirements, all test cases are chosen so that the memory is not depleted.

Both testing groups in the “Onion” testing suite require XS40, or XI50 appliance in the HW environment due to included security actions.

In both testing groups in the “Onion” testing suite, there are 4 dimensions which influence the observed measure throughput - concurrency, test case, testing scenario, and testing environment. Therefore, 5-dimensional graphs (4 dimensions plus one measure throughput)

would be ideal, however, not very practical for observing results on a 2-dimensional paper. Thus, in all graphs, the dimension testing environment is either fixed or added to some other dimension (because it has only 2-3 values) and one other dimension is always fixed. With such a restriction, we are constructing 3-dimensional graphs which can be projected to 2-dimensional paper. Admittedly, this lead us to many possible views on the measured throughput (permutation of 5 dimensions), not counted all possible values of the fixed dimension. Therefore, only the most illustrative combinations of dimensions and fixed values are shown, other views on the measured throughput can be built according to the Subsection 12.1.

8.2.1. Auction Testing Group

Figure 104 and Figure 105 provides a cross-scenario comparison of throughputs of the test case Auction_test0.002 (fixed dimension) in both environments. Consequently, Figure 106 depicts the cross-size comparison of throughputs of different test cases belonging to the Auction_XSLT testing scenario (fixed dimension).

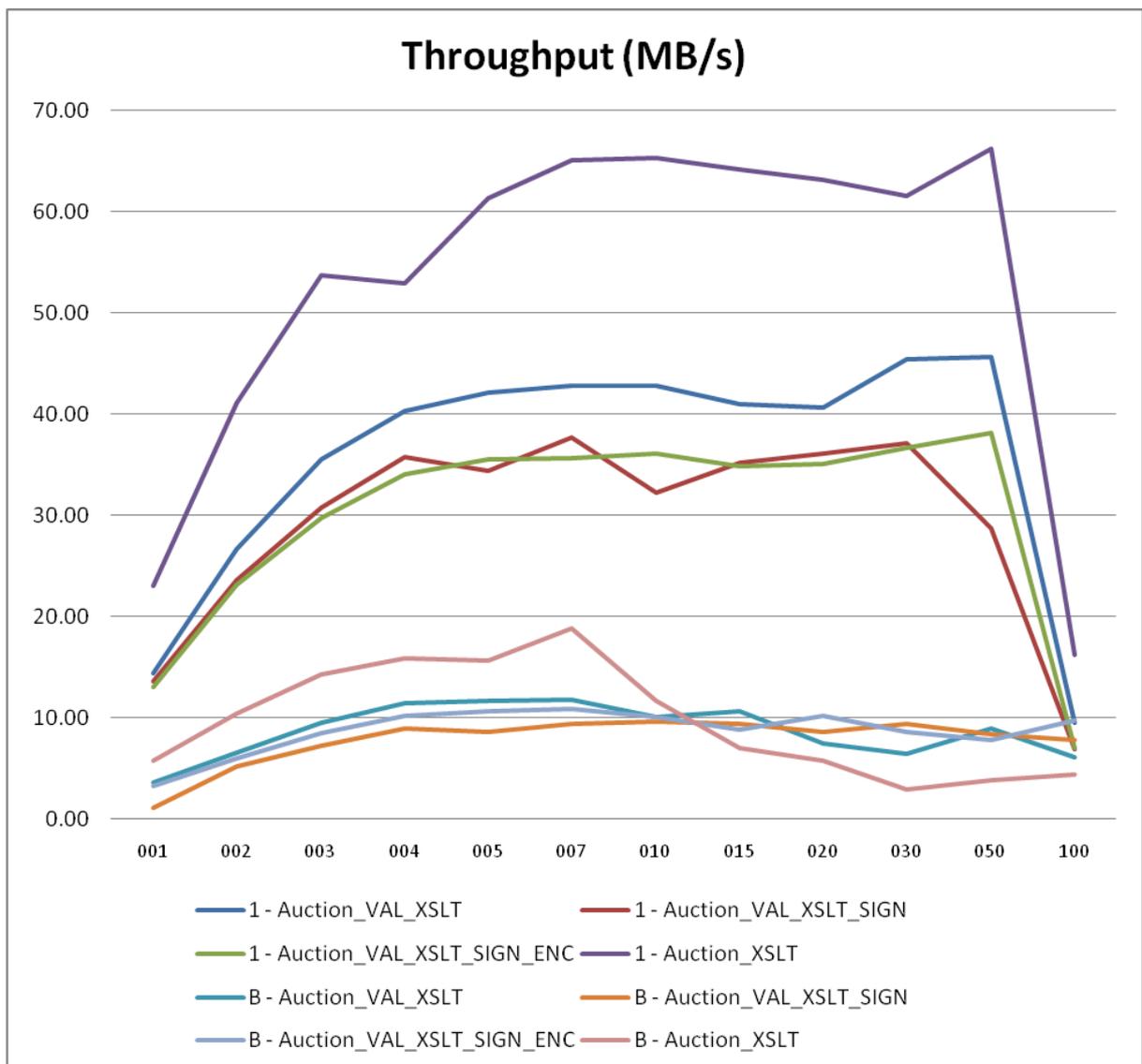


Figure 104 Cross-scenario Comparison of Test0.002 in Both Environments

In Figure 104, we can see the comparison of results in both environments at the small expense of lucidity. However, the four bottom lines on graph in Figure 104 concerning throughput in the SW environment are depicted in more detail in Figure 105.

Firstly, in the SW environment, CPU utilization is reaching 100% at the concurrency level 4, which corresponds with results of the “Flat” testing suite, where all tests typically fully depleted one core of the School server. In the HW environment, 100% workload is beginning on the concurrency levels 7-10 for Auction_XSLT testing scenario and falling down to concurrency levels 4-5 for other testing scenarios.

The second important fact, which yields from the previous affirmation, is the lack of support for XSL transformations in the SW environment, where the average throughput of Auction_XSLT testing scenario is approximately 6x lower than in the HW environment. To be more precise, the medium to tough transformation in Auction_XSLT testing scenario applied to rather small testing data of size 210 KB yields in less than 10% usage of the network, not counting in all other actions applied in other more complex testing scenarios of the Auction testing group.

In the HW environment, the throughputs of the Auction_VAL_XSLT_SIGN and Auction_VAL_XSLT_SIGN_ENC testing scenarios are similar due to relatively small outputs of the XSLT transformation and due to security accelerators on XI50 appliance.

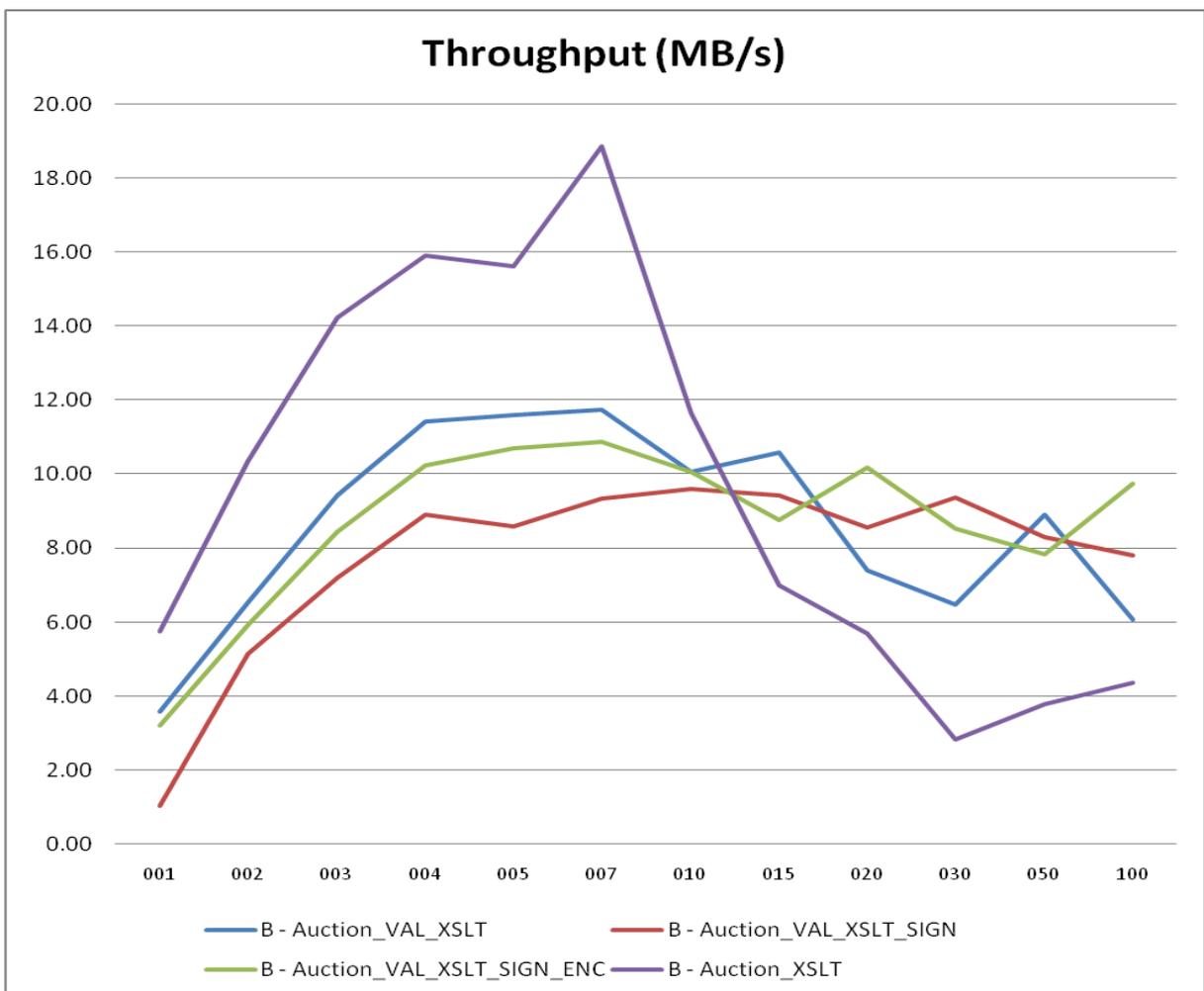


Figure 105 Cross-scenario Comparison of Test0.002 in the SW Environment

Figure 106 depicts comparison of throughputs of all test cases of the Auction_XSLT testing scenario in HW (left side) and SW (right side) environment.

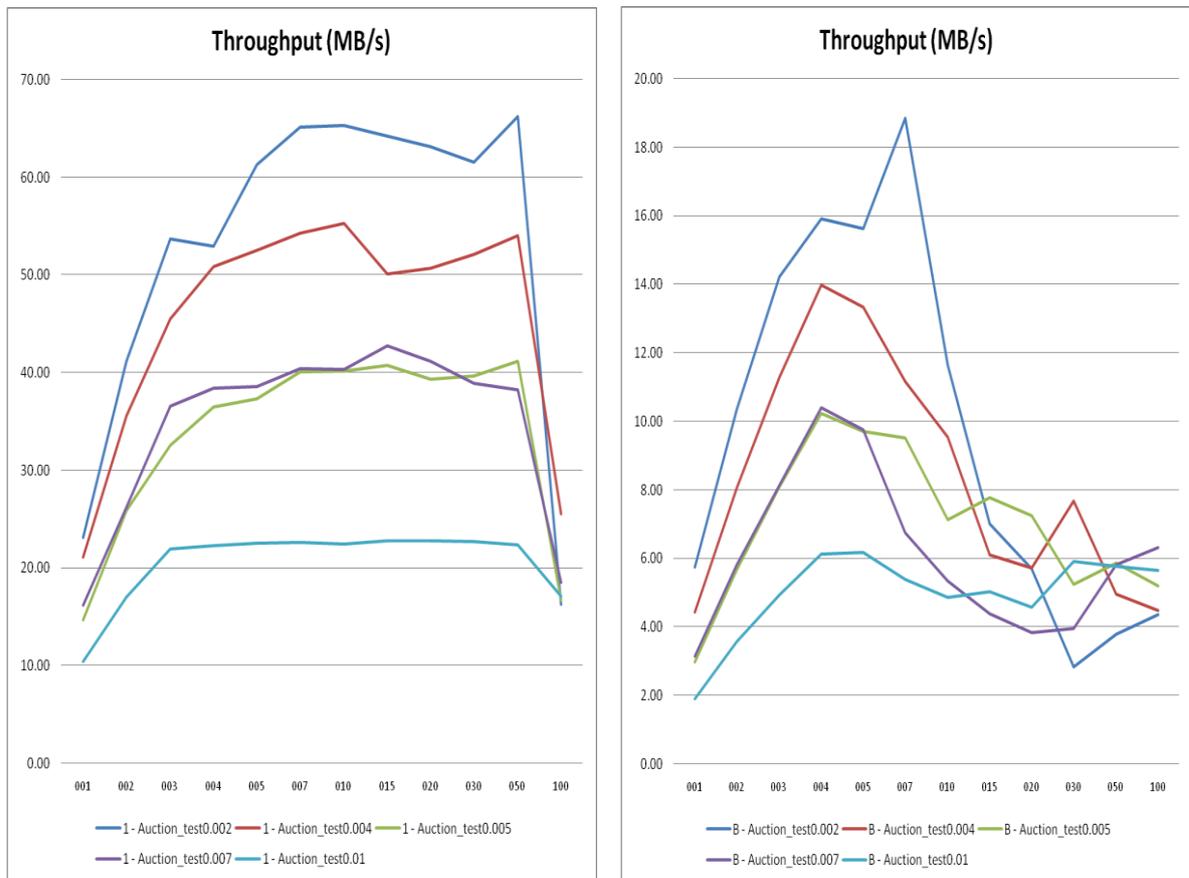


Figure 106 Cross-size comparison of Test Cases of Auction_XSLT Testing Scenario

As we can see, the maximum throughput for Auction_test0.002 (the test case with the smallest testing data) is in the HW environment 5x higher than in the SW environment and affirms very good support for XSLT transformations even if the input testing data are rather small (210KBs).

In the SW environment, the point of optimal throughput is very unstable for test cases with smaller testing data (test0.002, test0.004) and falls down with a small increase or decrease in concurrency level. Moreover, the optimal throughput concurrency level is shifting to the smaller values with increasing size of input testing data (concurrency level 7 for test0.002, concurrency level 5 for test0.004). In the HW environment, the optimal or almost optimal throughput is stable, for wide range of concurrency levels (beginning with the concurrency level 5 and ending with concurrency level 50), which gives better presumption about the total processing power of a group of HW appliances as well as about the number of HW appliances needful for particular XML processing task.

The resulting graphs for other testing scenarios of the Auction testing group can be viewed in the attached reports (see Subsection 12.1).

Finally, Figure 107 purveys descending trend of throughputs with the growing size of the testing data and growing complexity of the testing scenarios in HW (on the left side) and SW (on the right) environment. Concurrency level is fixed to one.

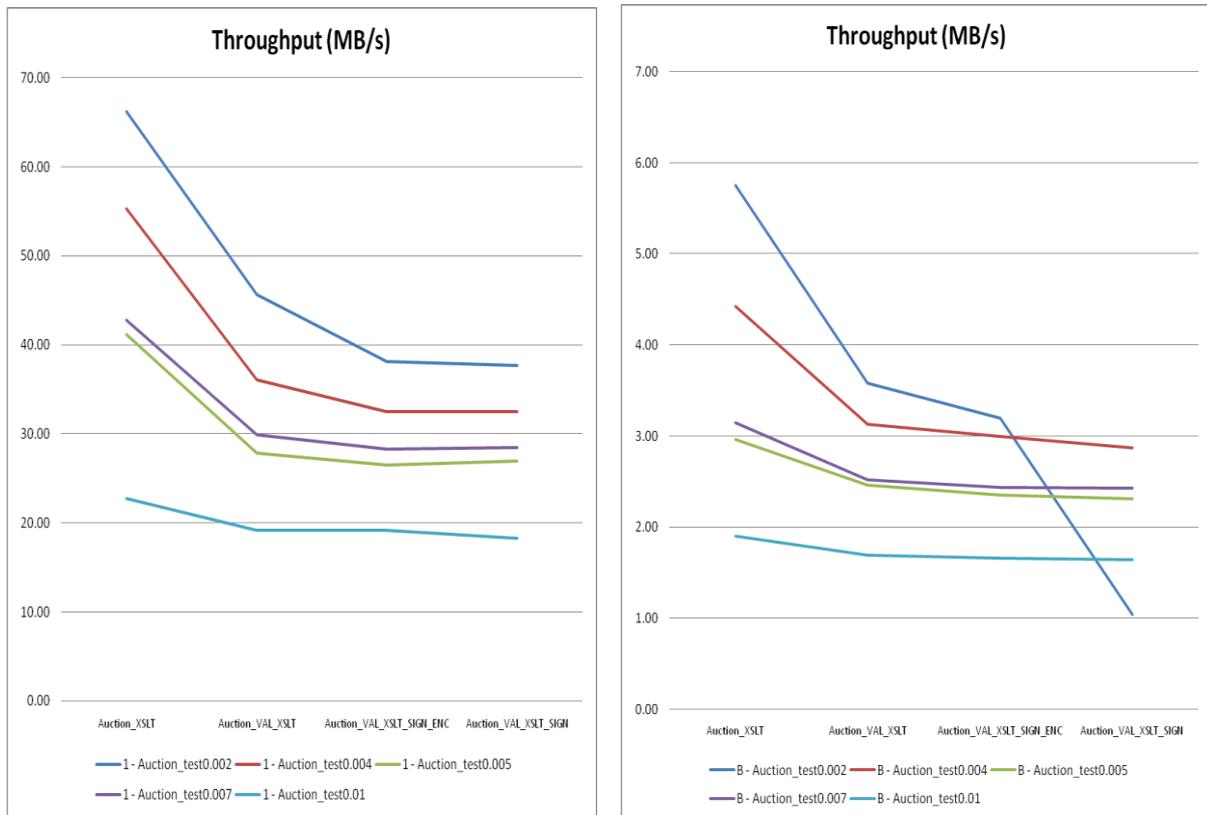


Figure 107 Cross-Scenario and Cross-size Comparison for C=1 in HW and SW

8.2.2. CSVOutput Testing Group

Figure 108 and Figure 109 purvey a cross-scenario comparison of throughputs of the test case CSV_rows200 (fixed dimension) in HW and SW testing environments.

The keywords “Piped” and “NonPiped” depicted in the graphs after the name of the testing scenarios of the CSVOutput testing group are just indicating, whether the PIPED context is used where possible or not. However, “NonPiped” keyword is used only in case of CSV_XSLT testing scenario, where it is equal to the “Piped” variant. Therefore, in the following graphs, all testing scenarios in the CSVOutput testing group are using PIPED contexts where possible.

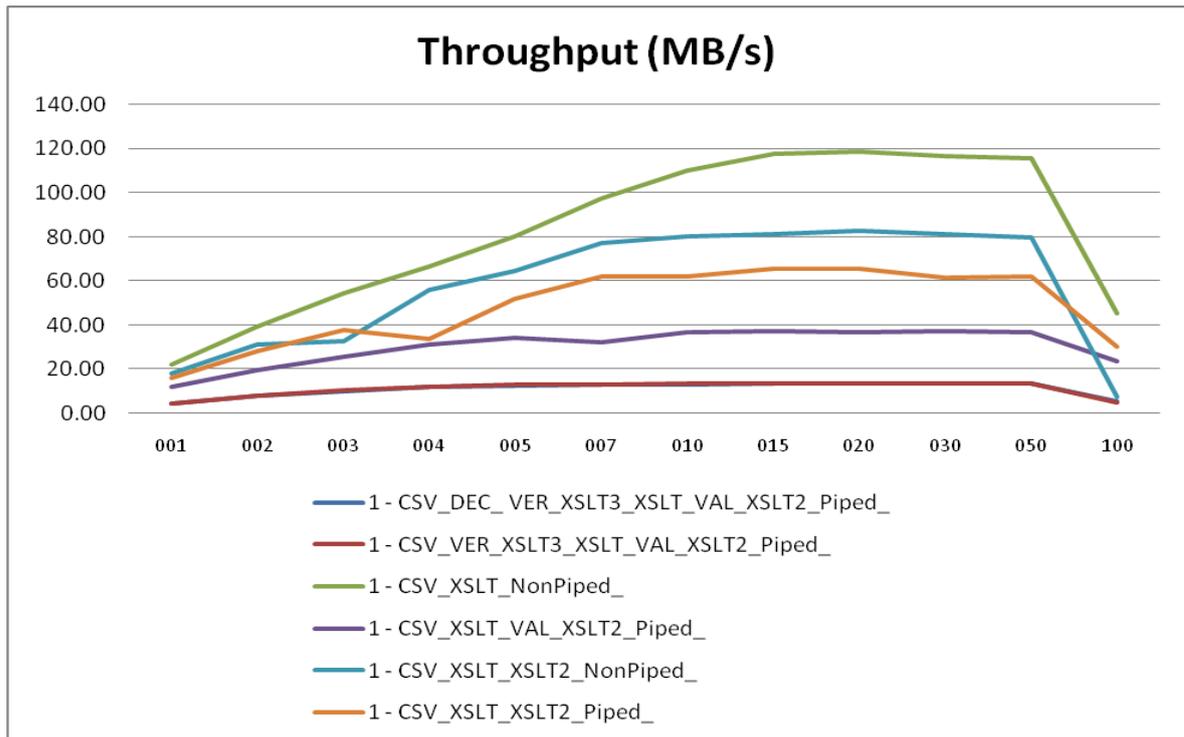


Figure 108 Cross-scenario Throughput for Rows200 Test Case in HW

In the HW environment, the throughput is culminating for CSV_XSLT testing scenario for concurrency levels 15-30, where the utilization of the network line is about 94%. After adding the second XSL transformation, the throughput almost halved, when the XSL transformations are PIPED (the yellow curve), however, if the named context is used to connect two XSL transformations (the turquoise-coloured curve), the throughput is about 30% higher in comparison with the PIPED one. This is little bit surprising, because the gain of the usage of the PIPED context does not prove for any of the four different sizes of the testing data used within the CSVOutput testing group, except for very large concurrency levels. However, for security reasons relating with the better memory utilization, the PIPED context is preferred in a real-world testing environment.

The results of the testing scenarios containing just Verifying action (red curve) and both Verifying and Decrypting actions (blue curve) are similar, the red line is hiding the blue one. The utilization of the network of the most complex testing scenario is only about 15%.

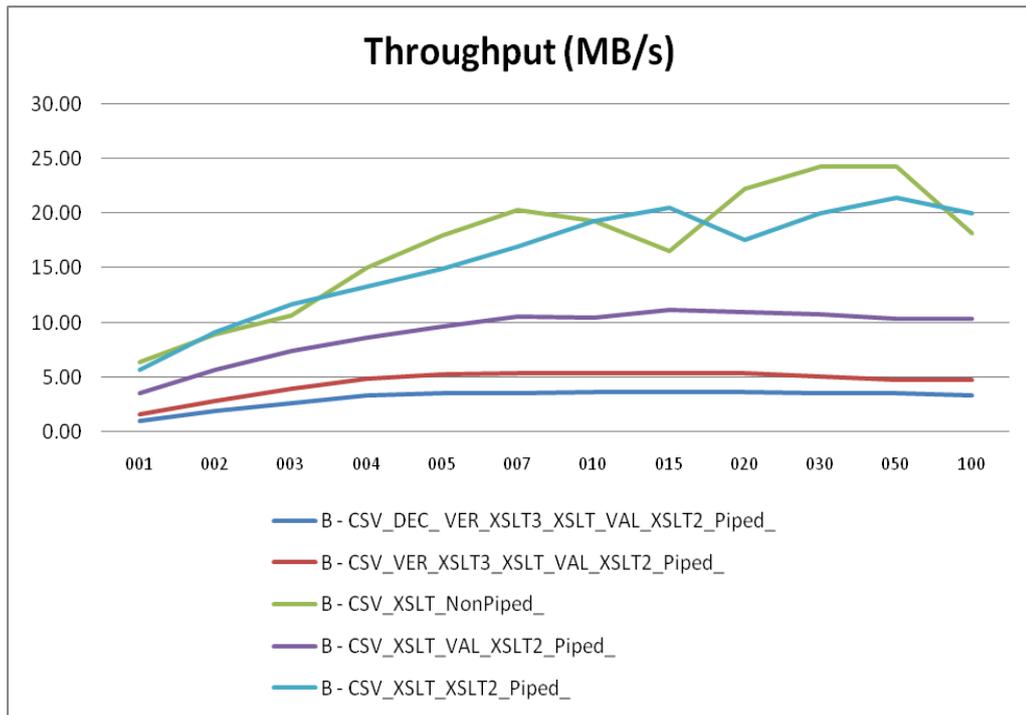


Figure 109 Cross-scenario Throughput for Rows200 Test Case in SW

In the SW environment, the throughputs of CSV_XSLT and CSV_XSLT_XSLT2 testing scenarios are similar, with the highest values for concurrency levels 30-50. Nevertheless, the network utilization is still no more than 19% of 1Gb bandwidth.

Figure 110 depicts a cross-scenario comparison of throughputs of the test case CSV_rows2000 (fixed dimension) in the SW environment, instead of using CSV_rows200 as in Figure 109.

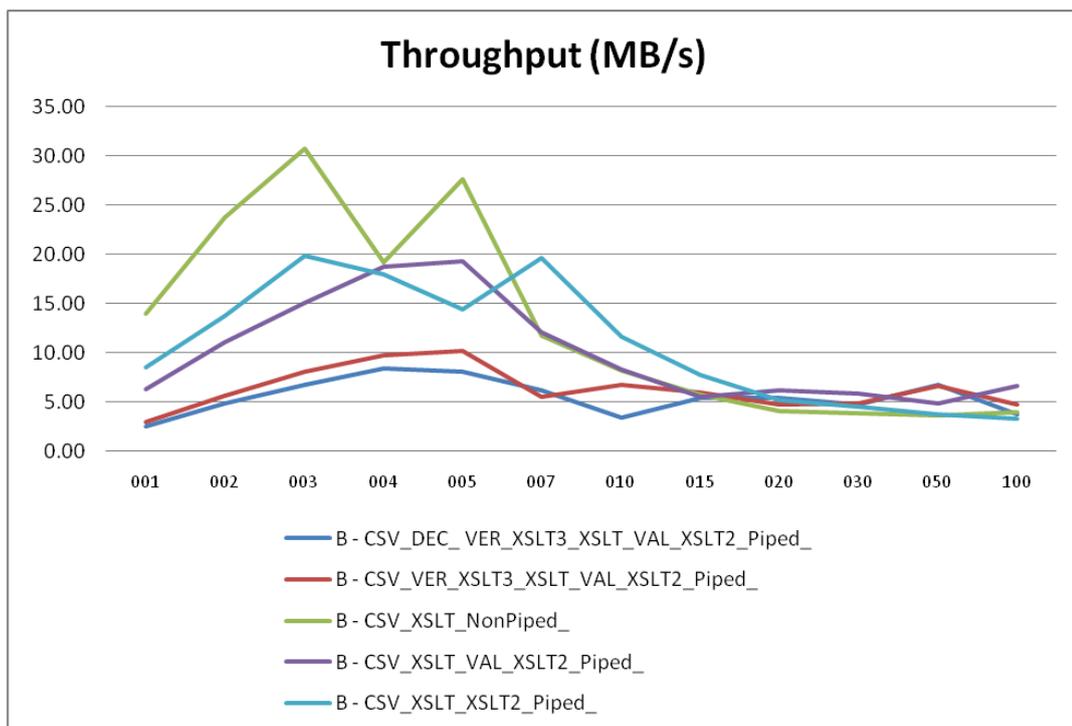


Figure 110 Cross-scenario Throughput for Rows2000 Test Case in SW

The throughput of all testing scenarios is rapidly falling down for higher levels of concurrency (10 and more). On the other hand, in the HW environment, the shapes of the curves are similar to Figure 108 with even higher measured throughput values due to involving test case with larger testing data and, thus, lower network overhead.

Figure 111 holds comparison of throughputs of test cases CSV_rows20, CSV_rows200, and CSV_rows2000 of the CSV_XSLT testing scenario in both testing environments.

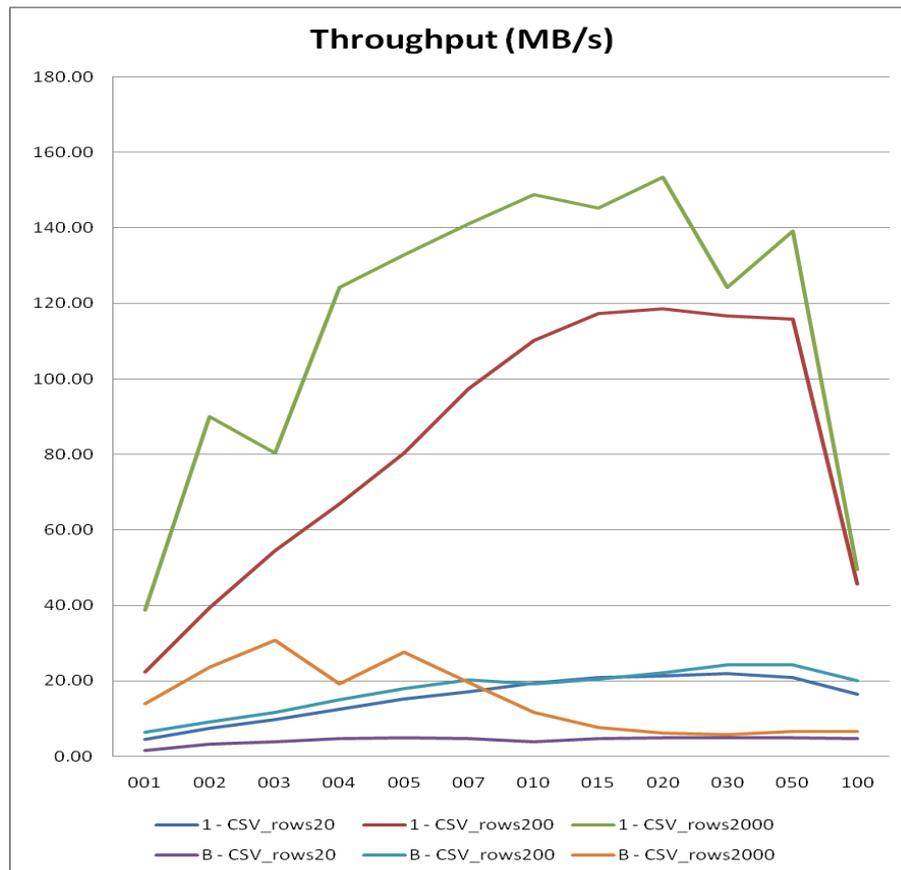


Figure 111 Cross-size Throughput of CSV_XSLT Testing Scenario in HW and SW

In the HW environment, the highest throughput is achieved for CSV_rows2000 test case, followed by CSV_rows200 test case, and CSV_rows20. In the SW environment, CSV_rows2000 test case has the highest throughput only for lower concurrency levels, for concurrency level higher than seven, CSV_rows200 wins and CSV_rows2000 loses rapidly as is illustrated in Figure 110 as well.

Last but not least, Figure 111 and Figure 112 depict the descending trend of throughputs with the growing size of testing data and growing complexity of the testing scenarios in HW (on the left side) and SW (on the right) environments. Concurrency level is fixed to one.

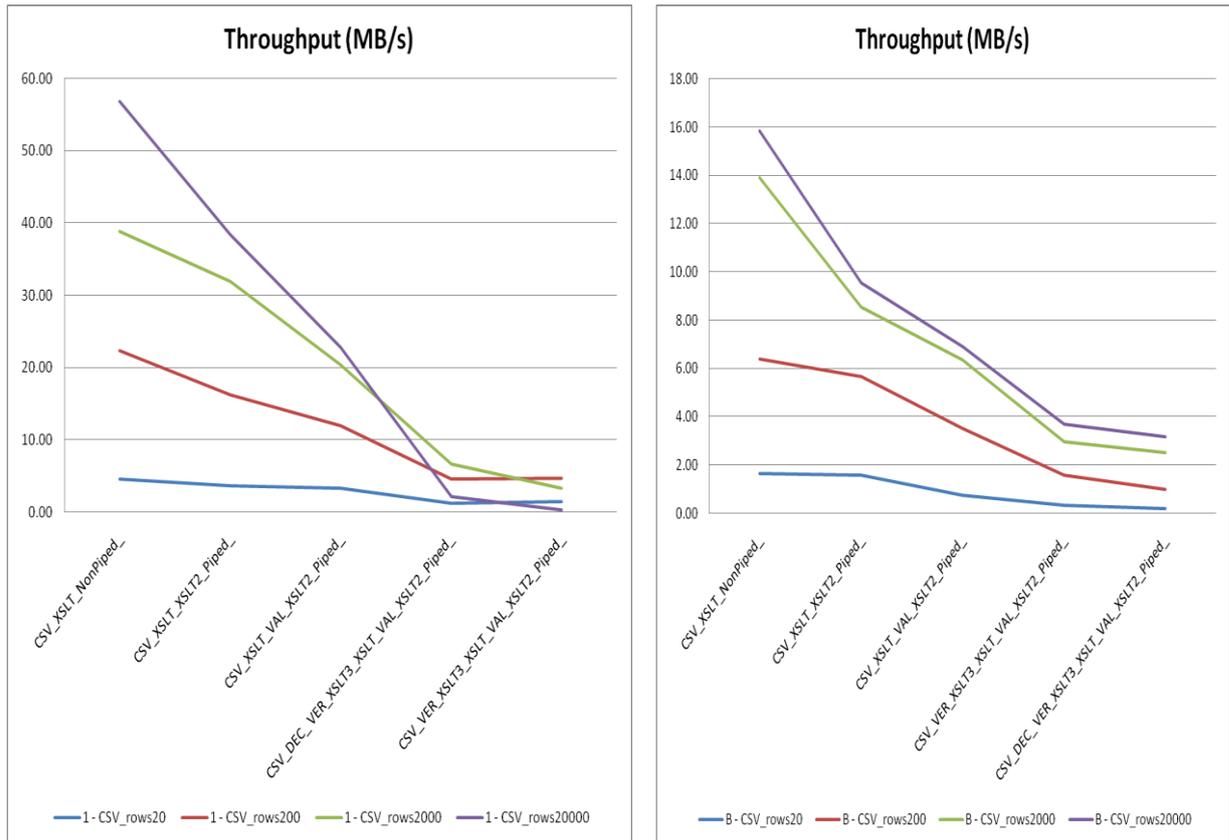


Figure 112 Cross-Scenario and Cross-size Comparison for C=1 in HW and SW

8.2.1. Cross-scenario Comparison

Finally, let us have look on percentage utilization of 1Gb network by various chosen testing scenarios. For each testing scenario, the maximum average test case throughput with a particular concurrency level is counted.

Figure 113 shows the collected results. Network usage less than 10% is emphasized by red colour and network utilization over 50% is marked by green colour. For each testing scenario, the percentage utilization is depicted, together with the test case name and concurrency level for which the maximum throughput is measured. Figure 114/Figure 115 depicts the percentage throughputs in comparison with 1Gb network in both environments of all chosen testing scenarios sorted according to reached results in the HW/SW environment.

Testing Scenario	SW - % of 1Gb	HW-% of 1Gb
Auction_XSLT	9.9 (c7, Auction_test0.002)	51.7 (c50, Auction_test0.002)
Auction_VAL_XSLT	6.1 (c20, Auction_test0.002)	35.6 (c50, Auction_test0.002)
Auction_VAL_XSLT_SIGN	9.6 (c10, Auction_test0.002)	29.4 (c7, Auction_test0.002)
Auction_VAL_XSLT_SIGN_ENCRYPT	8.5 (c7, Auction_test0.002)	29.8 (c50, Auction_test0.002)
CSV_XSLT	25.8 (c15, CSV_rows20000)	119.9 (c20, CSV_rows2000)
CSV_XSLT_XSLT2	16.0 (c15, CSV_rows200)	64.3 (c50, CSV_rows2000)

CSV_XSLT_VAL_XSLT2	15.2 (c5, CSV_rows20000)	39.8 (c15, CSV_rows20000)
CSV_VER_XSLT3_XSLT_VAL_XSLT2	8.0 (c5, CSV_rows2000)	10.5 (c20, CSV_rows2000)
CSV_DEC_VER_XSLT3_XSLT_VAL_XSLT2	6.6 (c4, CSV_rows2000)	11.4 (c20, CSV_rows2000)

Figure 113 Network Utilization of the “Onion” Testing Suite with Optimal Test Cases

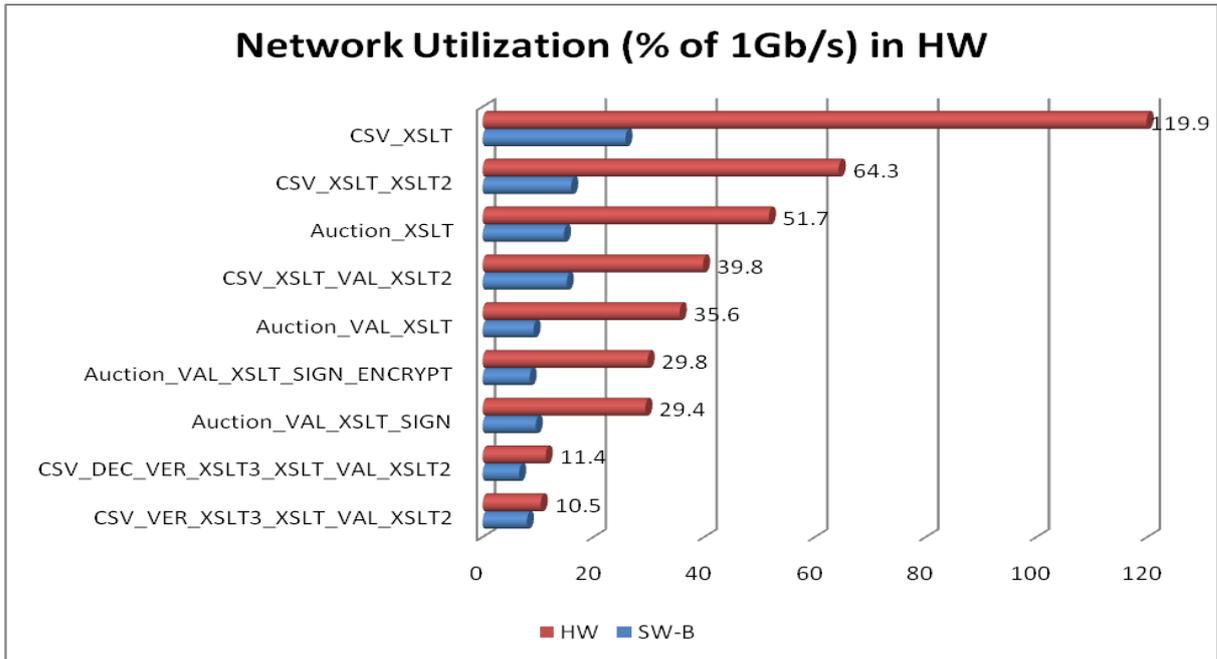


Figure 114 Network Utilization of the “Onion” Testing Suite in HW

In the HW environment, the CSV_XSLT testing scenario is dominating with reaching 119% usage of 1Gb network. Therefore, if the size of the testing data, the concurrency level, and the complexity of the XSL stylesheet are appropriate, the wire-speed processing of XML documents is real. The second rank of the scale belong to CSV_XSLT_XSLT2, which is predictable because of far more easier XSL stylesheets utilized in CSVOutput testing group than in Auction testing group. Nevertheless, the testing scenario Auction_XSLT with tough XSL stylesheet reaches better results than the testing scenario CSV_XSLT_VAL_XSLT2 containing two easy XSL stylesheets and easy XML Schema validation on the level of the testing scenario Val_XSD_REDUCED. Verifying and Decrypting actions achieve worse maximum throughputs than Encrypting and Signing actions.

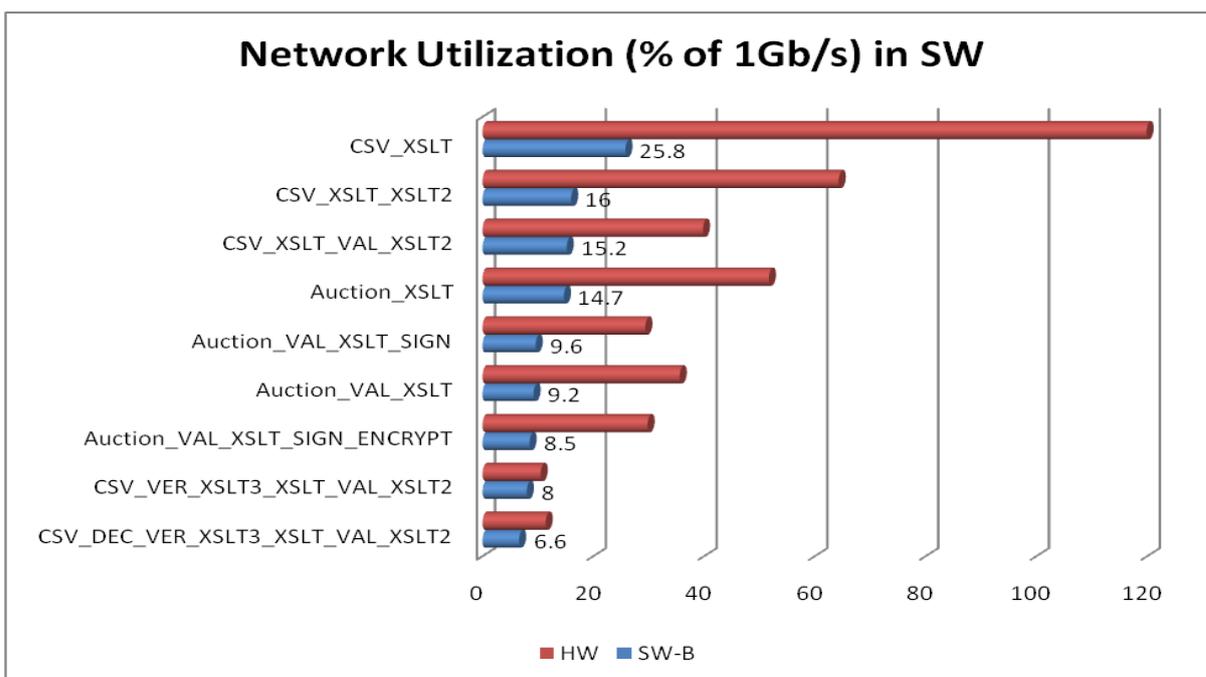


Figure 115 Network Utilization of the “Onion” Testing Suite in SW

In the SW environment, the same two testing scenarios win, however only with maximum utilization 25.8% and 16%, which is rather low for such easy XSL stylesheets utilized by these testing scenarios. In contrary to the HW environment, the testing scenario CSV_XSLT_VAL_XSLT2 with two easy XSL stylesheets and easy XML Schema validation achieves higher network utilization than Auction_XSLT containing just one, however, tough XSL transformation. Hence, Figure 115 underscores the non-effectiveness of XSL transformations in the SW environment.

V. RESULTS, CONCLUSION, AND FUTURE WORK

9. CONCLUSION

The aim of this work was to compare XML processing abilities of standard software environments (SW environments) and hardware accelerated environments (HW environments).

To start with, we have defined a testing framework which consists of a client and server parts utilizing XI50 appliance and WebSphere Application Server as representatives of HW and SW environments, respectively. To add, in the SW environment, we have designed and implemented a servlet-based J2EE application deployable on WebSphere Application Server with logic for parsing, validating, transforming, and securing XML data. Furthermore, the application enables definition of processing policies in a similar way as on XI50 appliance and is capable of caching auxiliary testing files (such as XSL stylesheets) and metadata about test cases.

After that, we have specified the testing hierarchy consisting of two testing suites, six testing groups, tens of testing scenarios, hundreds to thousands of test cases and tens of thousands of test case runs containing different testing data and utilizing different subsets of XML actions.

The testing hierarchy was executed on our testing framework and gathered measures were collected and analyzed using n-dimensional OLAP cubes. We produced reports containing all gathered and computed measures aggregated to all levels of testing hierarchy, starting with the average values of a particular measure for a test case and ending with the average values of a particular measure for the testing suite. Consequently, graphs showing the throughputs of the test cases from one testing group as well as across more testing groups were constructed and analyzed. Moreover, the sorted scale of testing scenarios according to its maximal throughput was built and summary results were formulated.

The test establishes, that HW accelerated XML processing brings 3.36x/4.42x higher average throughput measured over all test cases of all testing scenarios and all testing groups in the “Flat”/“Onion” testing suite. The difference in the throughput is most noticeable in case of XSL transformations. In Transforming testing group, the average throughput is in the HW environment 15x higher than in the SW environment measured over all test cases of all testing scenarios in the Transforming testing group and over all tested engines in both environments.

The “Onion” testing suite demonstrates that the wire-speed XML processing is reachable, if the size of the testing data, the concurrency level, and the complexity of XSL stylesheets are suitable. Moreover, in the HW environment, the optimal or almost optimal throughput lasts for a wider range of concurrency levels than in the SW environment, thus, the throughput value is much more stable and predictable.

Nevertheless, the performance gain relating to the connection of the HW appliance to the business network is not automatic and could be achieved only if several preconditions are observed. Firstly, we have to find out, that the processing of XML is the real bottleneck of the process we would like to speed up and, what is more, we must ensure that all other subprocesses of the accelerated process can prepare data for and consume data from the new XML processing appliance according to the estimated throughput of the data.

Secondly, XI50 should be considered as a message-oriented, not batch-oriented appliance. In other words, be aware of sending too large files (hundreds of megabytes) over the network, or at least try to connect the XML processing appliance in a proxy mode, so that one extra network hop is omitted, because the overhead of sending large files over the network using TCP/IP stack is high. Not mention the fact, that in case of non-streamable data, XI50 has a limited amount of memory and if the device is almost out of memory and cannot be revitalised by a garbage collection, it is silently rebooted and temporarily unavailable for tens of seconds. The reasonable limits on the size of XML documents depend on the complexity of the processing policy. On the other hand, the streaming approach can process even very large files, however, only Parsing, Validating and a subset of Transforming actions are streamable. And, what is more, streaming of large XML documents can be potentially very dangerous. The problem is what should be done, if a processing error occurs at the end of 100GB file, when the majority of the file is already out of the appliance.

Another important aspect is the economic point of view. Let us assume that the preconditions for incorporating HW appliance to the business network are satisfied and the sextuple acceleration of the XML processing can be expected due to suitable XSL transformations. Consequently, we can exchange six common servers with one XI50 appliance. If we possess 30 common servers, we can expect to buy just 5 XI50 appliances (emergency and testing servers as well as XI50 appliances are not counted). Perhaps, 4 or 6 appliances would be necessary, depending on the efficiency of the load balancers, network topology etc. Still, this leads us to significant savings in depleted electric energy and, what is more, we could probably dismiss one or two full-time administrators of these servers. Annual gross income of two senior administrators would be at least comparable to the price of one XI50 appliance.

If we consider directly our situation, let us assume that our School server costs 15,000\$ with all SW installed on it and XI50 with XG4 accelerator card costs 90,000\$. Again, if the throughput is in the HW environment 6x higher than in the SW environment (which is realistic in XSL transforming testing scenarios), we have the same price per 1Mb/s of throughput. Moreover, the question of administration, power, and room savings is still not taken into account.

To sum up, if certain preconditions are taken into account, the hardware appliance can bring significant acceleration to processed XML data.

10. FUTURE WORK

Further work can involve conformance testing to clarify, if the XML processing appliance is fully satisfying the used XML standards. For example, Extensible Markup Language Conformance Test [W71] can be executed.

Furthermore, the tests can be launched on HW appliances from different vendors and/or on the farm of servers and hardware appliances to verify the behaviour of more than one appliance. However, the appliances are hardly available in Czech Republic and, therefore, we have to wait until the situations meliorates.

Moreover, the results of processing SOAP messages can be gathered and analysed. Although the SOAP messages are XML data, they have their specific characteristics. Besides, WS-Security standard can be exploited and its performance compared with the performance of securing raw XML data.

Ultimately, XI50 has a great advantage in memory addressable by XPath queries. Hence, we can go even further, try to modify the memory management of some open source kernel and compare the gathered results with the results measured on XI50 appliance.

VI. REFERENCES AND APPENDICES

11. REFERENCES

[Erl] Thomas Erl: Service-Oriented Architecture, A Field Guide to Integrating XML and Web Services, Prentice Hall, 2004 (379-415)

[Joh] Rod Johnson: Expert One-on-One J2EE Design and Development, Wiley Publishing, 2003 (15-40, 203-248)

[Man] Sal Mangano: XSLT Cookbook, O'Reilly, 2002 (149-200)

[McG] James McGovern, Sameer Tyagi, Michael E. Stevens, Sunil Mathew: Java Web Services Architecture, Morgan Kaufmann Publishers, 2003 (277-312)

[Mly] Mlýnková, I.: XML Benchmarking: Limitations and Opportunities. Technical report 2008/1. Charles University, Prague, Czech Republic, January 2008

[Ste] Christopher Steel, Ramesh Nagappan, Ray Lai: Core Security Patterns - Best Practises and Strategies for J2EE, Web Services and Identity Management, Prentice Hall, 2006 (297-325)

[Szy] Clemens Szyperski, Dominik Gruntz, Stephan Murer : Component Software, Second Edition, Addison-Wesley, 2005

[W1] <http://www.ibm.com>,
Homepage of International Business Machines Corporation (IBM)

[W2] <http://www-306.ibm.com/software/websphere/>,
Information about IBM WebSphere software (including IBM WebSphere Application Server)

[W3] <http://en.wikipedia.org>,
Homepage of English version of free electronic encyclopaedia

[W4] <http://www-306.ibm.com/software/integration/datapower/>,
Homepage of IBM WebSphere DataPower SOA Appliances (including XI50 appliance)

[W6] <http://www.altova.com/>,
Homepage of XML utilities produced by Altova

[W7] <http://curl.haxx.se/>,
Homepage of curl utility for sending HTTP requests

[W8] <http://eclipse.org>,
Homepage of Java Integrated Development Environment (IDE)

[W9] <http://www.w3.org/XML/>,
Outline of XML standards

[W10] <http://www.cisco.com/en/US/products/ps7314/index.html>,
XML processing appliance made by Cisco

[W12] <http://www.layer7tech.com/products/page.html?id=71>,
XML processing appliance produced by Layer 7 Technologies

[W14] <http://www.contivo.com/>,
Homepage of Contivo, a company which interests in semantic integration of data

[W15] http://www-306.ibm.com/common/ssi/rep_ca/3/897/ENUS106-253/ENUS106253.PDF, IBM WebSphere DataPower Announcement, March 2006

[W16] <http://www.w3.org/TR/xmlsig-core/>,
XML Signature 1.0 Syntax and Processing, W3C Recommendation, February 2002

[W17] <http://www.itl.nist.gov/fipspubs/fip180-1.htm>,
Secure Hash Standard Specification

[W18] <http://www.itl.nist.gov/fipspubs/fip186.htm>,
Digital Signature Standard

[W19] <http://www.w3.org/TR/xmlenc-core/>,
XML Encryption 1.0 Syntax and Processing, W3C Recommendation, December 2002

[W20] <http://jcp.org/en/jsr/detail?id=105>,
JSR 105, XML Digital Signature Java API, Final Release, June 2005

[W21] <http://jcp.org/en/jsr/detail?id=106>,
JSR 106, XML Digital Encryption API, Proposed Public Review Draft, December 2005

[W26] <http://jcp.org/en/jsr/detail?id=206>,
JSR 206, Java API for XML Processing (JAXP) 1.3, Maintenance Release, November 2006

[W28] <http://www.ietf.org/rfc/rfc2396.txt>,
RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax

[W29] <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>,
Data Encryption Standard

[W30] <http://www.rsa.com/rsalabs/node.asp?id=2125>,
RSA Cryptographic Standard

[W31] <http://www.xml.com/pub/a/2001/03/28/xsltmark/index.html>,
XSLT Benchmark

[W32] <http://www.codesynthesis.com/projects/xsdbench/>,
XML Schema Benchmark

[W33] <http://httpd.apache.org/docs/2.0/programs/ab.html>,
Documentation of Apache HTTP server benchmarking tool (AB tool)

[W34] <http://www.w3schools.com/dtd/default.asp>,
Document Type Definition tutorial

[W35] <http://www.w3.org/MarkUp/SGML/>,
An overview of Standard Generalized Markup Language resources

[W37] <http://www.w3.org/XML/Schema>,
XML Schema on W3C

[W38] <http://www-306.ibm.com/software/info/education/assistant/>,
IBM Educational Assistant

[W39] <http://www.w3.org/TR/DOM-Level-3-Core/>,
W3C DOM Level 3 Specification, April 2004

[W40] <http://www.saxproject.org/>,
SAX Project

[W41] <http://jcp.org/en/jsr/overview>,
Java Community Process Overview

[W42] <http://xerces.apache.org/xerces2-j/>,
Homepage of Apache Xerces2 Parser

[W43] <http://xml.apache.org/crimson/>,
Homepage of Apache Crimson Parser

[W44] <http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1/index.html>,
Homepage of MXP1 - XML Pull Parser

[W45] <http://www.xmlpull.org/>,
Homepage of API for pull parsing

[W46] <http://www.dom4j.org/>,
Open source library dom4j for working with XML, XPath and XSLT

[W47] <http://santuario.apache.org/>,
Apache XML Security Project

[W48] <http://monetdb.cwi.nl/xml/>,
XMark - An XML Benchmark Project

[W50] <http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html>,
JSR 154, Java Servlet 2.5 Specification, Final Release

[W51] <http://jcp.org/aboutJava/communityprocess/final/jsr244/index.html>,
JSR 244, Java Platform, Enterprise Edition 5 Specification

[W52] <http://java.sun.com/javaee/technologies/>,
Java EE Technology

[W53] http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf,
Java 2 Platform, Enterprise Edition Specification, v1.4

[W54] <http://www.w3.org/>,
Homepage of World Wide Web Consortium

[W55] <http://www.w3.org/TR/xpath>,
XML Path Language 1.0, W3C Recommendation, November 1999

[W56] <http://www.w3.org/TR/REC-xml/>,
Extensible Markup Language 1.0 (Fourth Edition), W3C Recommendation, August 2006

[W57] <http://relaxng.org/>,
Relax NG Homepage

[W58] <http://xml.ascc.net/resource/schematron/schematron.html>,
Schematron, an XML structure validation language

[W59] <http://www.w3.org/TR/xslt>,
XSL Transformations, Version 1.0, November 1999

[W60] <http://www.ietf.org/rfc/rfc2818.txt>,
RFC 2818, HTTP over TLS

[W61] <http://jcp.org/en/jsr/detail?id=005>,
JSR 005, XML Parsing Specification, Final Release, March 2000

[W64] <http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic/server>,
BEA WebLogic Server

[W65] <http://www.jboss.com/products/jbossas>,
JBoss Enterprise Application Platform

[W66] <http://www.sun.com/software/products/appsrvr/index.jsp>,
Sun GlassFish Enterprise Server

[W67] <http://www.w3.org/TR/xquery/>,
XQuery 1.0, W3C Recommendation, January 2007

[W70] <http://www.lumrix.net/dtd2xs.php>,
Homepage of dtd2xs for converting DTD files to XSD files

[W71] <http://www.w3.org/XML/Test/>,
Extensible Markup Language (XML) Conformance Test Suites

[W72] <http://tools.ietf.org/html/rfc3280>,
RFC 3280, Internet X.509 Public Key Infrastructure, Certificate and Certificate Revocation List (CRL) Profile

[W73] <http://jcp.org/aboutJava/communityprocess/final/jsr914/index.html>,
JSR 914, Java Message Service (JMS) API, Maintenance Release, March 2002

[W74] <http://www-306.ibm.com/software/integration/wmq/>,
Homepage of IBM WebSphere MQ

[W75] <http://xml.coverpages.org/saml.html>,
Security Assertion Markup Language (SAML), Technology Report

[W76] <http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-os.html>, WS-SecureConversation 1.3, OASIS Standard, March 2007

[W77] <http://www.w3.org/TR/soap/>,
Simple Object Access Protocol (SOAP) 1.2, W3C Recommendation, April 2007

[W78] <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>, Web Services Security 1.1, OASIS Standard, February 2006

[W79] <http://web.mit.edu/Kerberos/>,
Kerberos, The Network Authentication Protocol

[W80] <http://tools.ietf.org/html/rfc2865>,
RFC 2865, Remote Authentication Dial In User Service (RADIUS)

[W81] <http://tools.ietf.org/html/rfc4510>,
RFC 4510, Lightweight Directory Access Protocol (LDAP)

[W82] <http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.htm>,
Web Service Distributed Management (WSDM), OASIS Standard, August 2006

[W83] <http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf>,
Advanced Encryption Standard (AES), FIPS 197, November 2001

12. APPENDICES

The following subsections contain supplements to this thesis.

12.1. Appendix A - Contents of the Enclosed DVD-ROM

This work is accompanied by a DVD-ROM with the following folder structure (sorted alphabetically):

- Core J2EE Application - Involves core J2EE application used in the sever part of the SW testing framework, where it implements the logic of processing policies and contexts. It is included as a deployable J2EE EAR file or as a folder tree export from IBM Rational Application Developer.
- OLAP Reports - Embraces the summary OLAP reports and several OLAP cubes for individual slicing and dicing of the cube according to the attached manual. Thanks to OLAP cubes, all results and all gathered measures can be viewed in a pleasant way.
- Testing Data - Holds testing data used in both testing environments
- Testing Scripts - Contains scripts for batch testing in HW as well as in SW environment
- Tools - Other tools created or downloaded and used for the purpose of the thesis

12.2. Appendix B – Important Java Packages

The following chapters contain important Java packages used when developing the application for processing XML requests in the SW environment. The lists of Java packages are adopted from particular Java documentations.

12.2.1. Java API for XML Processing 1.3 - JSR 206 [W26]

javax.xml	Core XML constants and functionality from the XML specifications.
javax.xml.datatype	XML/Java Type Mappings.
javax.xml.namespace	XML Namespace processing.
javax.xml.parsers	Provides classes allowing the processing of XML documents.
javax.xml.transform	This package defines the generic APIs for processing transformation instructions, and performing a transformation from source to result.
javax.xml.transform.dom	This package implements DOM-specific transformation APIs.
javax.xml.transform.sax	This package implements SAX2-specific transformation APIs.
javax.xml.transform.stream	This package implements stream- and URI- specific transformation APIs.
javax.xml.validation	This package provides an API for validation of XML documents.
javax.xml.xpath	This package provides an object-model neutral API for

	the evaluation of XPath expressions and access to the evaluation environment.
org.w3c.dom.*	Provides the interfaces for the Document Object Model (DOM) which is a component API of the Java API for XML Processing
org.xml.sax	This package provides the core SAX APIs.
org.xml.sax.ext	This package contains interfaces to SAX2 facilities that conformant SAX drivers won't necessarily support.
org.xml.sax.helpers	This package contains "helper" classes, including support for bootstrapping SAX-based applications.

Figure 116 Packages of Java API for XML Processing

12.2.2. XML Digital Signature API for Java 1.0 - JSR 105 [W20]

javax.xml.crypto	Common classes for XML cryptography.
javax.xml.crypto.dom	DOM-specific classes for the javax.xml.crypto package.
javax.xml.crypto.dsig	Classes for generating and validating XML digital signatures.
javax.xml.crypto.dsig.dom	DOM-specific classes for the javax.xml.crypto.dsig package.
javax.xml.crypto.dsig.keyinfo	Classes for parsing and processing KeyInfo elements and structures.
javax.xml.crypto.dsig.spec	Parameter classes for XML digital signatures.

Figure 117 Packages of XML Digital Signature API for Java

12.2.3. XML Digital Encryption API for Java 1.0 - JSR 106 [W21]

javax.xml.crypto.enc	Classes for parsing, encrypting and decrypting XML EncryptedType structures.
javax.xml.crypto.enc.dom	DOM-specific classes for the javax.xml.crypto.enc package.
javax.xml.crypto.enc.keyinfo	Classes for parsing and processing KeyInfo elements and structures.
javax.xml.crypto.enc.spec	Parameter classes for XML Encryption.

Figure 118 Packages of XML Digital Encryption API for Java