
This is the accepted version of the paper.

This version of the publication may differ from the final published version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/19833/

Link to published version:

Copyright: City Research Online aims to make research outputs of City, University of London available to a wider audience. Copyright and Moral Rights remain with the author(s) and/or copyright holders. URLs from City Research Online may be freely distributed and linked to.

Reuse: Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.
Self-Reconfigurable, Intrusion-Tolerant, Web-Service Composition Framework

by

Shahedeh Abdolhossein Khani

Submitted in fulfilment of the requirement for the degree of

Doctor of Philosophy

in the

City, University of London

School of Mathematics, Computer Science and Engineering

Department of Computer Science

MAY 2018
# Table of Contents

List of Figures ........................................................................................................... 6
List of Tables ............................................................................................................... 9
List of Abbreviations ................................................................................................. 10
Acknowledgements ..................................................................................................... 11
Abstract ...................................................................................................................... 12
Chapter 1 Introduction ............................................................................................... 14
  1.1 Motivation ............................................................................................................ 15
  1.2 Context ............................................................................................................... 17
  1.3 Research Questions ............................................................................................. 18
  1.4 Contributions ...................................................................................................... 19
  1.5 Research Distinctions .......................................................................................... 19
  1.6 Outline ............................................................................................................... 19
  1.7 Publication ......................................................................................................... 21
Chapter 2 Background and Motivation ...................................................................... 22
  2.1 Web Services Overview ...................................................................................... 23
    2.1.1 Web Services’ Core Technologies .............................................................. 25
    2.1.2 Summary .................................................................................................... 27
  2.2 Web Services’ Security Challenges and Issues .................................................. 28
    2.2.1 Security Vulnerabilities Concerning WSs .................................................. 30
    2.2.2 Existing Countermeasures and their Limitations ....................................... 36
    2.2.3 Summary .................................................................................................... 42
  2.3 Dependability and Intrusion-Tolerance ............................................................. 43
    2.3.1 Fault-Prevention ......................................................................................... 43
    2.3.2 Fault-Tolerance ......................................................................................... 44
    2.3.3 Fault-Removal ......................................................................................... 48
    2.3.4 Fault-Forecasting .................................................................................... 49
    2.3.5 Intrusion-Tolerance .................................................................................. 49
    2.3.6 Summary .................................................................................................... 49
  2.4 Penetration Testing ............................................................................................. 50
  2.5 Principal Component and Cluster Analysis ....................................................... 51
    2.5.1 Cluster Analysis ....................................................................................... 51
    2.5.2 Principal Component Analysis ................................................................... 53
### 2.5.3 Application of PCA in Cluster Analysis

2.5.4 Summary

### 2.6 WS Orchestration and Choreography

2.6.1 WSCI

2.6.2 BPML

2.6.3 BPEL

2.6.4 Summary

### 2.7 Summary

Chapter 3 Architecture of Reconfigurable ITWS framework

3.1 Objective

3.2 Assumptions

3.3 Architecture

3.4 Summary

Chapter 4 ITWS Formed Based on Penetration Test Results of Candidate WSs...

4.1 Penetration Testing Tool, WS-Attacker

4.1.1 WS-Attacker Framework

4.1.2 WS-Attacker Plugin

4.2 Case Study: Feasibility of ITWS Formed Based on Penetration Test Results of Candidate WSs

4.2.1 WSs Preparation

4.2.2 Penetration Tests Settings

4.2.3 WSs’ Penetration Test Results

4.2.4 ITWS Implementation

4.2.5 ITWS's Penetration Test Results

4.3 Summary

Chapter 5 Effects of BPEL on WSs’ XML-Related Security Vulnerabilities

5.1 Case Study: Effects of BPEL on WSs’ XML-Related Security Vulnerabilities

5.1.1 WS Preparation

5.1.2 WS Wrapped within BPEL process

5.1.3 Penetration Tests Settings

5.1.4 Attack Elements for Test SOAP Messages

5.1.5 Penetration Tests Results

5.2 Summary

Chapter 6 Security-Aware Selection of Optimal Group of WSs using PCA and CA, for Implementation of TWS
6.1 Case Study: Security-Aware Selection of Optimal Group of WSs using PCA and CA, for Implementation of ITWS .......................................................... 90
   6.1.1 WS Preparation .................................................................................. 91
   6.1.2 WSs Wrapped within BPEL processes ............................................ 91
   6.1.3 Penetration Tests Settings................................................................. 92
   6.1.4 Attack Elements for Test SOAP Messages ...................................... 93
   6.1.5 Penetration Test Results of Candidate WSs .................................... 93
   6.1.6 Principal Components Analysis of Candidate WS’s Penetration Test Results .................................................................................................................. 106
   6.1.7 Cluster Analysis based on Principal Component Analysis Results .... 114
   6.1.8 WS-Groups Ordering using Penetration Testing .............................. 116
6.2 Summary.................................................................................................. 122

Chapter 7 Dynamic Reconfiguration of ITWS Using BPEL and JAVA as BPEL Extension .......................................................... 123
   7.1 Setups for Case Studies.......................................................................... 123
      7.1.1 WS Preparation ............................................................................. 123
      7.1.2 DB Preparation ............................................................................. 124
      7.1.3 Communication with DB ............................................................... 125
   7.2: Case Study: Dynamic Reconfiguration Using a Combination of Java as BPEL Extension and BPEL Constructs ........................................................... 125
   7.3: Case Study: Dynamic Reconfiguration Using Only Java as BPEL Extension .............................................................................................................. 130
   7.4: Summary................................................................................................ 134

Chapter 8 Evaluation .................................................................................... 135
   8.1 Advantages and Limitations of the Presented ITWS ......................... 136
      8.1.1 Advantages of the Presented ITWS .............................................. 136
      8.1.2 Extensibility of the Presented ITWS ............................................ 136
      8.1.3 Limitations of the Presented ITWS .............................................. 137
   8.2 Feasibility of Implementing ITWS Based on Penetration Testing Results of Candidate WSs .......................................................... 137
   8.3 Feasibility of PCA and CA Utilization in Security-Aware Service Selection .......................................................... 138
   8.4 Feasibility of Implementing Self-Reconfigurable ITWS Using BPEL and JAVA as BPEL Extension .......................................................... 143

Chapter 9 Literature Review ........................................................................ 151
   9.1 Intrusion Detection, Prevention and Tolerant Systems ......................... 151
      9.1.1 Related Security Standards ............................................................ 151
9.1.2 Related ID/IP Approaches ................................................................. 152
9.1.3 Related Intrusion-Tolerant Systems .................................................. 153
9.2 Service Selection ................................................................................. 154
9.3 Reconfiguration/Adaptation ................................................................. 155

Chapter 10 Conclusions and Future Work .................................................. 157
10.1 Summary ............................................................................................ 157
10.2 Future Work ....................................................................................... 159
10.3 Conclusions ....................................................................................... 159

References .................................................................................................. 161

Appendix A: WSDL of Axis1-4 WS running on apache-tomcat-6.0.18 server .... 169
Appendix B: WSDL of Axis2-1.5.1 WS running on apache-tomcat-6.0.18 server . 172
Appendix C: WSDL of Axis2-1.6.1 WS running on apache-tomcat-6.0.18 server. 176
Appendix D: WSDL of CXF-2.5.11 WS running on apache-tomcat-7.0.72 server. 180
Appendix E: WSDL of CXF-2.3.10 WS running on apache-tomcat-6.0.18 server. 182
Appendix F: WSDL of CXF-2.6.3 WS running on apache-tomcat-7.0.72 server ... 184
Appendix G: Java Class for Communication between ITWS and Database ......... 186
Appendix H: Java Class for Dynamic WS Invocation Using RPC Library .......... 188
Appendix I: Java Class for Dynamic WS Invocation through RPC and Java Multi-
Threading Libraries ....................................................................................... 191
Appendix J: Remaining Java Codes for Chapter 7 ........................................... 194
List of Figures

Figure 2.1: The Service Oriented Operational Architecture [1] ........................................... 24
Figure 2.2: Simple WS Interaction [1] ............................................................................ 25
Figure 2.3: Discrete Components in a WS Architecture [1] ............................................. 26
Figure 2.4: Composite WS [1] .......................................................................................... 27
Figure 2.5: UT Model [23] ................................................................................................. 29
Figure 2.6: Security Vulnerabilities Affecting SOA (implemented using WSs technology) .......................................................... 30
Figure 2.7: Web services Security Standards – National Reference Model [46] .... 37
Figure 2.8: The Dependability Tree [56] ......................................................................... 43
Figure 2.9: Sequential Alternative Pattern [63] .............................................................. 46
Figure 2.10: Parallel Selection Pattern [63] ................................................................... 47
Figure 2.11: Parallel Evaluation Pattern [63] ................................................................. 47
Figure 2.12: PCA in CA [102] .......................................................................................... 54
Figure 2.13: WSCI Collaboration [107] .......................................................................... 56
Figure 2.14: BPEL4WS Process Flow [107] .................................................................... 57
Figure 3.1: Overview of the Framework’s Objective ....................................................... 63
Figure 3.2: General Architecture, the dotted lines indicate the external systems .... 64
Figure 3.3: Combined SM and SGM Operation Intervals. The solid lines and dotted lines indicate SM and SGM operation intervals, respectively ........................................... 69
Figure 4.1: Overview of WS-Attacker and Its Processing Steps [18] ............................ 73
Figure 4.2: Component Diagram for WS-Attacker ........................................................... 73
Figure 4.3: BPEL Diagram of the ITWS .......................................................................... 80
Figure 5.1: BPEL Process for Wrapping the Developed WSs. ................................. 83
Figure 5.2: The Component Diagram for Direct Penetration Testing of the .......... 86
Figure 5.3: The Component Diagram for Penetration Testing of the WS in this Case Study while it is wrapped in a BPEL Process ......................................................... 86
Figure 5.4: Results from Direct Penetration Testing Axis2-1.6.1 WS (for information about each test see Table 5.1) ............................... 88
Figure 5.5: Penetration Test Results for Axis2-1.6.1 WS while it was wrapped in a BPEL Process (for information about each test see Table 5.1) ................................................. 88
Figure 6.1: Penetration Tests Results for Tests 1-8 Performed on Axis1-4 WS (for information about each test and WS see Table 6.1 and Section 6.1.1, respectively) ................................................. 94
Figure 6.2: Penetration Tests Results for Tests 9-16 Performed on Axis1-4 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively.................. 95
Figure 6.3: Penetration Tests Results for Tests 1-8 Performed on Axis2-1.5.1 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively ............. 96
Figure 6.4: Penetration Tests Results for Tests 9-16 Performed on Axis2-1.5.1 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively ................................. 97
Figure 6.5: Penetration Tests Results for Tests 1-8 Performed on Axis2-1.6.1 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively 98
Figure 6.6: Penetration Tests Results for Tests 9-16 Performed on Axis2-1.6.1 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively ................................................................. 99
Figure 6.7: Penetration Tests Results for Tests 1-8 Performed on CXF-2.3.10 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively ................................................................. 100
Figure 6.8: Penetration Tests Results for Tests 9-16 Performed on CXF-2.3.10 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively ................................................................. 101
Figure 6.9: Penetration Tests Results for Tests 1-8 Performed on CXF-2.5.11 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively ................................................................. 102
Figure 6.10: Penetration Tests Results for Tests 9-16 Performed on CXF-2.5.11 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively ................................................................. 103
Figure 6.11: Penetration Tests Results for Tests 1-8 Performed on CXF-2.6.3 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively ................................................................. 104
Figure 6.12: Penetration Tests Results for Tests 9-16 Performed on CXF-2.6.3 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively ................................................................. 105
Figure 6.13: Plot of Candidate WSs’ Penetration Tests Results, to be used as PCA inputs (for information about each test and WS see Table 6.1 and Section 6.1.1, respectively) ................................................................. 107
Figure 6.14: Variance Explained by each PC ...................................................... 112
Figure 6.15: Scree Graph .................................................................................. 113
Figure 6.16: The WSs’ Original Penetration Test Results against the First Two PCs ................................................................. 113
Figure 6.17: Component Diagram for Penetration Testing Each Group of WSs ... 116
Figure 6.18: Penetration Test Results for WS-Group1 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively) ................................................................. 118
Figure 6.19: Penetration Test Results for WS-Group2 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively) ................................................................. 118
Figure 6.20: Penetration Test Results for WS-Group3 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively) ................................................................. 119
Figure 6.21: Penetration Test Results for WS-Group4 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively) ................................................................. 119
Figure 6.22: Penetration Test Results for WS-Group5 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively) ................................................................. 120
Figure 6.23: Penetration Test Results for WS-Group6 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively) ................................................................. 120
Figure 6.24: WS-Groups Sorted in Ascending Order According to their Penetration Test Results (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively) ................................................................. 121
Figure 7.1: BPEL Diagram of Dynamically Reconfigurable ITWS, Implemented Using a Combination of Java as BPEL Extension and BPEL Constructs .............. 126
Figure 7.2: Process Execution Time (ms) for Dynamic ITWS Implemented using Combination of Java as BPEL Extension and BPEL constructs (ran with three diverse WSs)...................................................................................................................... 129
Figure 7.3: Process Execution Time (ms) for Dynamic ITWS Implemented using Combination of Java as BPEL Extension and BPEL constructs (ran with five diverse WSs)...................................................................................................................... 129
Figure 7.4: BPEL Diagram of Dynamic Reconfigurable ITWS, Implemented Using a Combination of Java as BPEL Extension and BPEL Constructs................................................................. 130
Figure 7.5: Process Execution Time (ms) for Dynamic ITWS Implemented using Java as BPEL Extension only (ran with three diverse WSs).............................................................. 133
Figure 7.6: Process Execution Time (ms) for Dynamic ITWS Implemented using Java as BPEL Extension only (ran with five diverse WSs).............................................................. 133
Figure 8.1: Penetration Test Results for WS-Group7 (for information about each test see Table 6.12)......................................................................................................................................................... 139
Figure 8.2: Average of Penetration Test Results for WS-Groups1-6 against Average of Penetration Test Results for WS-Group7 ......................................................................................................................... 139
Figure 8.3: BPEL Diagram of Static ITWS Implemented for Evaluation Purpose.. 143
Figure 8.4: Process Execution Time (ms) for Static ITWS Implemented using BPEL Constructs only (ran with three diverse WSs from Section 7.1.1).................................................... 146
Figure 8.5: Process Execution Time (ms) for Static ITWS Implemented using BPEL Constructs only (ran with five diverse WSs from Section 7.1.1).................................................... 146
Figure 8.6: Average of Process Execution Times (ms) for Static ITWS against the Average of Process Execution Times (ms) for Dynamic ITWSs.............................................................. 147
Figure 8.7: Dynamic Reconfiguration Time (ms) for ITWS Implemented using Combination of Java and BPEL Constructs........................................................................................................ 149
Figure 8.8: Dynamic Reconfiguration Time (ms) for ITWS Implemented using Java only ..................................................................................................................................................... 149
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Services Repository (SR)</td>
<td>67</td>
</tr>
<tr>
<td>3.2</td>
<td>Penetration Tests Repository (PTR)</td>
<td>68</td>
</tr>
<tr>
<td>3.3</td>
<td>Services Penetration Tests Results Repository (SPTRR)</td>
<td>68</td>
</tr>
<tr>
<td>3.4</td>
<td>Service-Groups Repository (SGR)</td>
<td>68</td>
</tr>
<tr>
<td>3.5</td>
<td>Groups Penetration Tests Results Repository (GPTRR)</td>
<td>68</td>
</tr>
<tr>
<td>3.6</td>
<td>Failure Records Repository (FRR)</td>
<td>68</td>
</tr>
<tr>
<td>4.1</td>
<td>WS-Attacker’s DoS Attack Success Metrics [40]</td>
<td>77</td>
</tr>
<tr>
<td>4.2</td>
<td>WS-Attacker’s DoS Attack’s Effect Metrics on Third-Party Users [40]</td>
<td>78</td>
</tr>
<tr>
<td>4.3</td>
<td>Results of Testing DoS Attack Plugins on a Number of Web service</td>
<td>78</td>
</tr>
<tr>
<td>4.4</td>
<td>Frameworks [40]</td>
<td>78</td>
</tr>
<tr>
<td>4.5</td>
<td>Penetration Tests Results</td>
<td>79</td>
</tr>
<tr>
<td>5.1</td>
<td>WS-Attacker’s Settings</td>
<td>81</td>
</tr>
<tr>
<td>6.1</td>
<td>WS-Attacker’s Settings</td>
<td>84</td>
</tr>
<tr>
<td>6.2</td>
<td>WSs’ Penetration Test Results to be used as PCA input</td>
<td>92</td>
</tr>
<tr>
<td>6.3</td>
<td>Mean Matrix of the Penetration Test Results</td>
<td>108</td>
</tr>
<tr>
<td>6.4</td>
<td>Mean Adjusted Penetration Test Results</td>
<td>108</td>
</tr>
<tr>
<td>6.5</td>
<td>Covariance Matrix of the Mean Adjusted Penetration Test Results</td>
<td>109</td>
</tr>
<tr>
<td>6.6</td>
<td>Eigenvectors for the Covariance Matrix of the Mean Adjusted Penetration Test Results</td>
<td>109</td>
</tr>
<tr>
<td>6.7</td>
<td>Ordered Eigenvalues</td>
<td>110</td>
</tr>
<tr>
<td>6.8</td>
<td>Ordered Eigenvectors</td>
<td>110</td>
</tr>
<tr>
<td>6.9</td>
<td>Calculated Principal Components</td>
<td>110</td>
</tr>
<tr>
<td>6.10</td>
<td>Results of CA on selected PCs</td>
<td>115</td>
</tr>
<tr>
<td>6.11</td>
<td>Distance between Each WS and the Cluster Centres (in 1.0e+04 scales)</td>
<td>115</td>
</tr>
<tr>
<td>6.12</td>
<td>WS-Attacker’s Settings</td>
<td>117</td>
</tr>
<tr>
<td>7.1</td>
<td>Service Table Storing Necessary Information for Invoking Candidate WSs</td>
<td>124</td>
</tr>
<tr>
<td>8.1</td>
<td>2-Samples Tests Results for Coercive Parsing Attack</td>
<td>141</td>
</tr>
<tr>
<td>8.2</td>
<td>2-Samples Tests Results for Hash Collision Attack</td>
<td>141</td>
</tr>
<tr>
<td>8.3</td>
<td>Execution Time Overheads</td>
<td>147</td>
</tr>
<tr>
<td>8.4</td>
<td>2-Sample Tests Results for ITWS ran with three WSs</td>
<td>148</td>
</tr>
<tr>
<td>8.5</td>
<td>2-Sample Tests Results for ITWS ran with five WSs</td>
<td>148</td>
</tr>
</tbody>
</table>
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AOP</td>
<td>Aspect Oriented Programming</td>
</tr>
<tr>
<td>BP</td>
<td>Business Process</td>
</tr>
<tr>
<td>BPEL</td>
<td>Business Process Engineering Language</td>
</tr>
<tr>
<td>BPEL4WS</td>
<td>Business Process Execution Language for Web services</td>
</tr>
<tr>
<td>BPML</td>
<td>Business Process Modelling Language</td>
</tr>
<tr>
<td>CA</td>
<td>Cluster Analysis</td>
</tr>
<tr>
<td>DII</td>
<td>Dynamic Invocation Interface</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>FRR</td>
<td>Failure Records Repository</td>
</tr>
<tr>
<td>GPTRR</td>
<td>Groups Penetration Tests Results Repository</td>
</tr>
<tr>
<td>HashDoS</td>
<td>Hash Collision</td>
</tr>
<tr>
<td>ID</td>
<td>Intrusion Detection</td>
</tr>
<tr>
<td>IP</td>
<td>Intrusion Prevention</td>
</tr>
<tr>
<td>ITWS</td>
<td>Intrusion-Tolerant Web Service</td>
</tr>
<tr>
<td>NVD</td>
<td>National Vulnerability Database</td>
</tr>
<tr>
<td>OSVDB</td>
<td>Open Source Vulnerability Database</td>
</tr>
<tr>
<td>OTSC</td>
<td>Off-The-Shelf Components</td>
</tr>
<tr>
<td>OTWSs</td>
<td>Off-The-Shelf Web Services</td>
</tr>
<tr>
<td>OWASP</td>
<td>Open Web Application Security Project</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>PCs</td>
<td>Principal Components</td>
</tr>
<tr>
<td>PTR</td>
<td>Penetration Tests Repository</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality-of-Service</td>
</tr>
<tr>
<td>RPCs</td>
<td>Remote Procedure Calls</td>
</tr>
<tr>
<td>SAML</td>
<td>Security Assertion Markup Language</td>
</tr>
<tr>
<td>SAX</td>
<td>Simple API for XML</td>
</tr>
<tr>
<td>SB-WS</td>
<td>Stock Broker WS</td>
</tr>
<tr>
<td>SGM</td>
<td>Service-Groups Manager</td>
</tr>
<tr>
<td>SGR</td>
<td>Service-Groups Repository</td>
</tr>
<tr>
<td>SM</td>
<td>Services Manager</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SPTRR</td>
<td>Services Penetration Tests Results Repository</td>
</tr>
<tr>
<td>SQ-WS</td>
<td>Stock Quote WS</td>
</tr>
<tr>
<td>SR</td>
<td>Services Repository</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>WA</td>
<td>Web Application</td>
</tr>
<tr>
<td>WASC</td>
<td>Web Application Security Consortium</td>
</tr>
<tr>
<td>WSCI</td>
<td>Web Service Choreography Interface</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSFL</td>
<td>Web Services Flow Language</td>
</tr>
<tr>
<td>WSs</td>
<td>Web Services</td>
</tr>
<tr>
<td>XACML</td>
<td>Extensible Access Control Markup Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Definitions</td>
</tr>
</tbody>
</table>
Acknowledgements

I would like to express my gratitude to my two supervisors, Dr Cristina Gacek and Dr Peter Popov. I am very grateful to them for their invaluable inspiration on this dissertation and all their supports. As their student I have learnt so many things. I would like to thank my examiners Professor Stefano Russo and Dr Evangelia Kalyvianaki for their useful suggestions.

Thanks go to Stephanie Wilson, Dr Andrew Macfarlane, David Mallo-Ferrer and everyone in the Centre for Software Reliability (CSR) for being so friendly and supporting me throughout these years.

I am immensely grateful to my family for their love and support. Thank you!
Abstract

The Internet has provided an opportunity for businesses to offer their services as Web Services (WSs). WSs are used to implement Service Oriented Architecture (SOA). They enable composition of independent services with complementary functionalities to produce value-added services, which results in less development effort, time consumption and cost, enabling companies and organizations to implement their core business only and outsource other service components over the Internet, either pre-selected or on-the-fly.

Simple Object Access Protocol (SOAP) based WSs are at risk of security vulnerabilities related to their specific implementation technologies such as Extensible Markup Language (XML) as well as those of their underlying platforms (e.g., operating systems and frameworks) and their applications (e.g., vulnerability to SQL Injection attacks). Cyber-attacks on WSs may cause unavailability, loss of confidentiality and/or integrity as well as significant monetary penalties. Security issues become more challenging when Off-The-Shelf Web Services (OTSWSs) are used since they are beyond the control of their clients.

The central question underlying this work is:

*Can a self-reconfigurable Intrusion-Tolerant Web Service, implemented using N-version programming and diversity formed by composing Off-The-Shelf Web Services that are selected through penetration testing, Principal Component Analysis, and Cluster Analysis processes mitigate XML-related security vulnerabilities?*

While aiming to answer the above question, this dissertation presents a novel framework to increase dependability by constructing an Intrusion-Tolerant Web Service (ITWS) in which *N-version programming and diversity, formed by composing SOAP-OTSWSs, is used. It describes how penetration testing can be used as a measure of security vulnerabilities of available SOAP-OTSWSs (that offer the required functionality) and the resultant ITWS,*
how Principal Component Analysis (PCA) and Cluster Analysis (CA) and be utilized to group the SOAP-OTSWSs based on their security vulnerabilities diversity and how a further penetration testing on each group of diverse SOAP-OTSWSs can be used to select the optimal set (most secure among the groups) for construction of ITWS.

This dissertation also demonstrates how the dynamic reconfiguration of ITWS, created in Business Process Engineering Language (BPEL), can be enabled using a combination of BPEL constructs and Java as BPEL extension approach and using only Java as BPEL extension approach.

The novelty of the work presented in this dissertation is twofold. On the one hand, it is security informed and on the other hand, it demonstrates the use of Java (as BPEL 2.0 extension) to implement self-reconfigurable composite WS. It has the advantage of, at the same time, facilitating a dependable service to users and exploiting existing standard technologies. This work also assesses the effectiveness of the proposed solutions through various case studies and discusses the implications of the proposed framework.
This dissertation is concerned with improving the dependability of Simple Object Access Protocol (SOAP) based Web Services (WSs) when Off-The-Shelf Web Services (OTWSWs) are employed. Its focus is on the security vulnerabilities related to WSs’ implementation technologies. It introduces a novel framework to increase dependability by constructing Intrusion-Tolerant Web Service (ITWS) in which N-version programming and diversity, formed by composing OTWSWs, is used. It also demonstrates how dynamic reconfiguration of ITWS can be enabled by using a combination of Business Process Engineering Language (BPEL) constructs and Java as BPEL extension approach and using only Java as BPEL extension approach.

This chapter starts by describing the motivation for the problem addressed by this dissertation (Section 1.1). It then presents the context of (Section 1.2) and the central question underlying (Section 1.3) this work. Afterward, it enumerates the contributions (Section 1.4) and lists the research distinctions (Section 1.5). Finally, it outlines the remaining chapters of this dissertation (Section 1.6) and presents its publication (Section 1.7).

1 The focus of this dissertation is on the SOAP-based WSs and the term “*WS”, wherever used in this dissertation, refers to this type of services.
1.1 Motivation

Service Oriented Architecture (SOA) is a popular paradigm for system integration and interoperation. It employs services as fundamental elements for developing applications. A service is a self-contained unit of software that provides a particular function. WSs are an implementation of the SOA. A WS can be as simple as a logging service or as complex as an organization’s business logic that is decided to be exposed to the outside world [1]. WSs can advertise their services (e.g., OTSWSs) through a registry. Hence, desirable WSs can be looked up (via their description) and used individually or integrated into a composite WS.

In WSs, all documents and data are in Extensible Markup Language (XML) format. This property has simplified the interoperability of the various technologies employed in the development of WSs [1]. It has also provided other attractive features such as simple request-response service exchange architecture (ease of use), platform independence, the ability to transport lots of information over the Internet and capability to compose loosely coupled components [1]. Because of these features, WSs are being employed in the development of various critical applications, such as banking, booking, e-business, e-science, etc. [2], [3]. Hence, ensuring their security has received considerable attention from researchers: security standards [4], authorization [5], access control [6], [7], anomaly detection [8], etc.

In addition to the above attractive features, XML has also introduced its specific security vulnerabilities that cannot be detected by Intrusion Detection and Intrusion Prevention systems. For example, a malformed message that does not comply with expected message structure can make the server unavailable or cause unintended operations. Use of WS-* security standards [4] and message validation (before they reach the business logic) are often cited as sufficient countermeasures to safeguard against attacks exploiting this type of security vulnerabilities [9]–[11].

The WS-* security standards are members of WSs specifications and apply integrity and confidentiality through SOAP extensions (e.g., XML Sig-
nature and XML Encryption). They also provide communication of several security token formats (e.g., Security Assertion Markup Language, Kerberos, and X.509). However, they have the following limitations:

- Equipping a SOAP message with these standards may require changes to the message’s structure since their use may introduce additional elements which did not exist previously.
- If their use is revoked, the message structure may require significant changes.
- They may cause security vulnerabilities themselves. For example, McIntosh and Austel [12] have shown that the content of a SOAP message, which is protected by an XML Signature (as specified in WS-Security standard) can get changed without invalidating the signature.

A SOAP message validator checks whether the message adheres to the specified schema and discards the message if it does not. However, this approach also has various limitations such as:

- Often such validators rely on XML Schema Definitions derived from Web Service Description Language document or hand-coded by programmers, which may make them prone to cyber-attacks.
- Similarly to the previous limitation, manual unguided schema updates that rely entirely on the skills of a programmer may also make the validator prone to cyber-attacks.
- If the schema used by the validator is loosely defined, it may allow malicious messages to pass through.
- They cannot safeguard against some cyber-attacks exploiting XML-related security vulnerabilities. For example, Jenson et al. [13] have shown that even the most restrictive XML Schema validators may fail to defend against XML Signature Wrapping attacks.

The security issue gets more challenging when OTSWSs are employed since they are ready-made black boxes of unknown quality and their security is out of the control of their clients. However, any attacks exploiting
their security vulnerabilities will also affect the services offered by their clients. Also, OTWSs may be updated (which may increase or decrease their security) or more secure OTWSs (offering the same required functionality) may become available. Therefore, their clients should be able to replace them to maintain or even improve the overall security, if they wish to. The replacing to a more secure OTWS may need to be done dynamically as switching off the client’s system might not be acceptable.

To address these challenges, a self-reconfigurable, intrusion-tolerant WS remains an interesting choice, which is the motivation for the work presented in this dissertation and one of its key benefits is the possibility of ensuring correct behaviour in the presence of attacks.

This dissertation presents a novel framework to increase dependability by constructing ITWSs in which N-version programming and diversity, formed by composing OTWSs, is used. It uses penetration testing as a measure of security vulnerabilities (XML-related) of available OTWSs (that offer the required functionality) and the resultant ITWS. It employs Principal Component Analysis (PCA) and Cluster Analysis (CA) to group the OTWSs based on their security vulnerabilities diversity then performs further penetration testing on each group to select the optimal set (most secure among the groups) for construction of ITWS. It also investigates the dynamic reconfiguration of ITWS, created in Business Process Engineering Language (BPEL), using a combination of BPEL constructs and Java as BPEL extension approach and using only Java as BPEL extension approach.

1.2 Context

This section discusses how the current work can be used.

Assume a Stock Broker WS (SB-WS), which uses a third-party Stock Quote WS (SQ-WS). If this SQ-WS becomes unavailable for any reason or returns incorrect stock prices, it will affect the SD-WS business and reputation. These problems can be diminished through redundancy approaches. For example, through SD-WS using SQ-WSs (offered by various vendors)
simultaneously to get the stock price then performing majority voting on the returned stock prices. In this way, SD-WS can continue to provide reliable service as long as the majority of the employed SQ-WSs return the correct stock price. This approach (redundancy) can also improve the overall security of the SD-WS if the utilized services have security vulnerabilities diversity. Otherwise, any common security vulnerabilities will be a window of opportunity for compromising all the SQ-WSs, having the targeted security vulnerability, at the same time that will also affect the SD-WS’s business and reputation. Also, SQ-WSs may become upgraded (which may increase or decrease their security) or more secure SQ-WSs may become available. Therefore, SD-WS should be able to perform dynamic (since switching off SD-WS will cause significant monetary penalties) self-reconfiguration to use the optimal set of SQ-WSs (set of SQ-WSs with most security vulnerabilities diversity).

The framework presented in this dissertation addresses the above needs of SD-WS as follows:

1. Redundancy: through a composite WS constructed using BPEL (a de facto language for WSs composition).
2. Security vulnerabilities diversity: by identifying the security vulnerabilities of each available SQ-WS through penetration testing then grouping these services based on their security vulnerabilities diversity using PCA and CA approaches and finally selecting the optimal set (most secure) through a further penetration testing.
3. Dynamic reconfiguration: through a combination of BPEL constructs and Java as BPEL 2.0 extension approach or using only Java as BPEL extension approach.

1.3 Research Questions

The central question underlying this work is:

*Can a self-reconfigurable ITWS, implemented using N-version programming and diversity formed by composing OTSWSs that are selected*
through penetration testing, PCA, and CA processes mitigate XML-related security vulnerabilities?

1.4 Contributions

In particular, this dissertation makes the following contributions:

1. A general architecture for an ITWS (formed by composing OTSWSs) that mitigates XML-related security vulnerabilities.
2. The integration of penetration testing, PCA, and CA to group OTSWSs based on their overall XML-related security vulnerabilities for ITWS implementation.
3. An approach to reconfiguring the ITWS (implemented in BPEL) using a combination of BPEL constructs and Java snippets (as BPEL 2.0 extension).
4. A method for reconfiguring the ITWS (implemented in BPEL) using only Java as BPEL 2.0 extension.

1.5 Research Distinctions

The major novelties of this work are as follows:

1. Consideration of XML-related security vulnerabilities in selecting OTSWSs and implementing ITWS.
2. Use of PCA and CA analysis on penetration tests results of OTSWSs in choosing a diverse group of services regarding their security vulnerabilities to implement ITWS.

1.6 Outline

The rest of this dissertation is organised as follows:

Chapter 2 further explains the motivation for the work of this thesis and describes its background. In particular, it elaborates on main areas that constitute the current context, namely, WSs’ architecture and characteristics, their security issues, their composition and self-reconfiguration capabilities.
and penetration testing them as well as dependability (*N-version programming* and *diversity*), *PCA* and *CA* approaches.

Chapter 3 explains the architecture of proposed framework and the assumptions made.

This dissertation demonstrates the proposed framework through some case studies. Chapter 4 introduces the penetration testing tool, utilized in the case studies presenting the proposed service selection framework. It then demonstrates the feasibility of *ITWS* implementation based on penetration test results of the candidate *WSs*.

Chapter 5 shows that *BPEL* could affect the *XML*-related security vulnerabilities of the candidate (for *ITWS* implementation) *WSs* and argues that these effects should be considered in service selection process.

Chapter 6 shows, how penetration test results of the candidate *WSs*, *PCA*, and *CA* could be used to group *WSs* based on their *XML*-related security vulnerabilities and how these groups could be sorted using further penetration testing.

Chapter 7 demonstrates the implementation of self-reconfigurable *WS* using a combination of *BPEL* constructs and *Java* as *BPEL* 2.0 extension approach and utilizing only *Java* as *BPEL* extension approach through two case studies.

Chapter 8 evaluates the work presented in this dissertation using the outcomes of the case studies covered in Chapters 4-7.

Chapter 9 reviews recent related work.

Finally, Chapter 10 provides a summary, conclusions, and discusses future work.
1.7 Publication

The following publication is based on this dissertation:

Chapter 2

Background and Motivation

Simple Object Access Protocol (SOAP) based Web Services (WSs) have attractive features such as simplified interoperability of the various technologies employed in their development. However, they may also have Extensible Markup Language (XML) related security vulnerabilities that can be mitigated through appropriate countermeasures such as the use of WS-* security standards and message validation (before a message reaches the business logic). But each of these approaches has its limitations especially when Off-The-Shelf Web Services (OTSWSs) are employed.

This chapter presents the background and motivation for this dissertation and identifies Intrusion-Tolerant Web Service (ITWS), created using N-version programming and diversity (formed by composing OTSWSs with security vulnerabilities diversity), as an interesting choice to safeguard against cyber-attacks exploiting WSs’ security vulnerabilities (including XML-related security vulnerabilities). In this work, a combination of penetration testing, Principal Component Analysis (PCA), and Cluster Analysis (CA) approaches are employed to select a set of most diverse OTSWSs regarding their security vulnerabilities and its motivation is given throughout this chapter.

In particular, this chapter is divided into the following sections, each corresponding to a different dimension that shapes this dissertation:
Section 2.1 provides an overview of the architecture and the characteristics of the WSs.

Section 2.2 provides an overview of security issues concerning WSs including their XML-related security vulnerabilities, the existing countermeasures to safeguard against cyber-attacks exploiting this type of security vulnerabilities and the limitations of such countermeasures. It then concludes that an ITWS remains an interesting choice to protect against cyber-attacks threatening WSs including those exploiting XML-related security vulnerabilities especially when OTSWSs are employed.

Section 2.3 presents a concise overview of the concepts and techniques of dependability and how intrusion-tolerance can be achieved using such methods.

Sections 2.4 and 2.5 briefly discusses penetration testing, PCA, and CA approaches and explains how they can be employed to achieve diversity (regarding security vulnerabilities) for ITWS purpose.

Section 2.6 provides an overview of WSs’ composition and how it can be used to implement ITWS. It then argues the usefulness of self-reconfigurable ITWS for security and reviews the capabilities and limitations of Business Process Engineering Language (BPEL) 2.0 in this matter.

Finally, Section 2.7 summarizes the discussions provided in this chapter and the motivations for the work presented in this dissertation.

2.1 Web Services Overview

A publically available service is a self-contained process (deployed over standard middleware such as J2EE) that can be described, published, discovered and invoked over a network [14]. It can be as simple as a logging service or as complex as an organization’s business logic (e.g., holiday booking service).

Service Oriented Architecture (SOA) is an architectural style for implementing software systems out of loosely coupled, interoperable services.
This approach supports the development of rapid, low-cost and easy to compose distributed applications [14]. An SOA typically consists of the following roles (the operational architecture of the SOA roles is shown in Figure 2.1):

- **Provider**: is the owner of the service that hosts its implementation, defines its description and publishes it to a registry [1].
- **Requester**: it can be either a person or another WS. It uses the service registry to identify its desirable service(s) through its (their) description then binds to it (them) using the binding information provided in its (their) description [1].
- **Registry**: it is a searchable registry of service descriptions (published by their owners) [16].

![Figure 2.1: The Service Oriented Operational Architecture [1]](image)

WSs are the standards-based realization of SOA [17]. They are modular, self-describing, self-contained software components, which are network accessible (e.g., they are accessible through HTTP) [14]. Each service consists of an implementation (which is accessible over a network) and an interface (which is based on XML). The interface contains the service’s description including its datatypes, operations, protocol bindings and the network location for its implementation (e.g., the URL for the service’s implementation) [1]. The characteristics of WSs include:

- **They are XML-based**: XML is a specification language and can be used to transmit or store data. In WSs, XML is used for invoking the
services and representing the data. Forasmuch as it can accommodate any data type and structure, it enables interoperability of the technologies adopted in the development of WSs [1], [18].

- **They are loosely coupled**: in WSs, the client and the server logics are loosely coupled. Therefore, any changes in either of these interfaces do not require an update in the other interface [1].

- **They are coarse-grained**: unlike object-oriented technologies that expose the services through several fine-grained methods, WSs only expose coarse-grained services [1].

- **They can be synchronous or asynchronous**: WSs can be either synchronous (the client should wait until the last service invocation is completed before invoking the next service) or asynchronous (the client can invoke different services concurrently) [1].

- **They support RPCs**: clients of the WSs can invoke procedures, functions, and methods on remote objects through an XML-based protocol [1].

- **They support document exchange**: in WSs, complex documents can be represented in XML format, which enables WSs to exchange them [1].

### 2.1.1 Web Services’ Core Technologies

![Figure 2.2: Simple WS Interaction](image)

SOAP [19], Web Service Description Language (WSDL) [20] and Universal Description, Discovery and Integration (UDDI) [21] are WSs’ major technologies [22], whose interaction is demonstrated in Figure 2.2.
SOAP

SOAP is a simple and lightweight protocol for exchanging the XML data over the Web [22]. It enables interoperability among the WSs’ heterogeneous clients and servers [1]. A SOAP message contains an Envelope (identifies the XML document as a SOAP message), a Header (contains Metadata such as timestamps) and a Body (contains call and response information) elements [1].

WSDL

WSDL is an XML language for describing a WS including its data type definitions, the operations it offers, its input/output message format, its network address, its protocol binding, etc. [22]. It assists the clients to understand how the interaction with the WS can be set.

UDDI

UDDI provides a registry where the WSs can be advertised and discovered by their names, specifications, etc. [1]. In the example presented in Figure 2.3, an application is acting as a client, which has submitted the information (e.g., specification) about its desirable services to a UDDI. The UDDI has identified the suitable services, based on the client’s requirements, and has returned the location of their WSDL to the client. The client is then
communicating with the identified services through SOAP messages and contact information provided in their WSDL files.

Sometimes the client may need to communicate with another client, which itself is communicating with a number of other WSs (see Figure 2.4) to perform a task. In this case, the target client and the WSs that it communicates with can be combined and encapsulated to form a composite service [1].

\[
\text{Figure 2.4: Composite WS [1]}
\]

### 2.1.2 Summary

This section briefly presented the architecture of the WSs and showed that they are modular, self-describing, self-contained software components, which are accessible through HTTP. These features allow companies to save on their costs by speeding the application(s) implementation and integration processes [23]. On the other hand, WSs also have security issues that have to be addressed. The next section first explains the concept of security properties, vulnerabilities, and attacks as well as security in distributed software systems. It then presents security vulnerabilities concerning WSs followed by an overview of a number of existing countermeasures against cyber-attacks exploiting WSs’ XML-related security vulnerabilities (as the focus of the work in this dissertation is on this type of security vulnerabilities). It then explains
some of the limitations of these countermeasures and concludes that an ITWS is a suitable countermeasure to safeguard against cyber-attacks threatening WSs including those exploiting XML-related security vulnerabilities, especially when OTSWSs are employed.

2.2 Web Services’ Security Challenges and Issues

Security Properties

The security objective of computer systems is to protect the stored information and the data that should be transferred over the networked devices. To achieve these objectives, the following main security properties should be satisfied [24]:

- **Authorization**: to ensure that users do not take action or access the resources and information beyond their specified rights.
- **Authenticity**: to ensure that the people or systems taking part in the communication are who they claim to be.
- **Confidentiality**: to ensure that the information will only be disclosed to the authorised people.
- **Integrity**: to ensure that the information will only be altered by the authorized people.
- **Non-repudiation**: to ensure that the involvement of the legitimate people or systems in a transaction cannot be denied.
- **Availability**: to ensure that authorized parties can access the information when needed.

Cyber-attacks and Vulnerabilities

Any action violating any of the above properties is an attack and any possibility (loophole in the security architecture of the computer system) enabling an attacker to harm the resources is a vulnerability [24]. Cyber-attacks can be divided into Passive and Active groups.

- **Passive attacks**: eavesdropping or accessing unauthorised data are the objectives of Passive attacks [23].
Active attacks: this type of attacks attempt to change the unauthorised data by making a modification or adding false data [23]. Active attacks can be further divided into the following groups:

- **Masquerade**: one entity tries to gain access to unauthorised data or resources by pretending to be an entity with this access right [23].
- **Replay**: replaying the message back to the sender as if it was the reply from the receiver [23].
- **Unauthorised access**: getting access and using unauthorised data/resources [23].
- **Unauthorised alteration**: illegitimate modification, removal or alteration of the data [23].
- **Repudiation of action**: a party denies an action it has performed [23].
- **Unauthorised DoS**: one party denies other entities authorised access to the resources [23].

Security in Distributed Software Systems

![UT Model](image)

Figure 2.5: UT Model [23]

Figure 2.5 illustrates the association between the security properties (explained previously) and a distributed software system such as SOA, implemented using WS technology. It is known as UT model [23], because it resembles the shape of the alphabet letters $U$ and $T$. The legs of the $U$ represent various layers of the distributed system and the $T$ demonstrates the se-
curity properties and their management that may be required across the layers of the distributed system [23].

2.2.1 Security Vulnerabilities Concerning WSs

SOA Security Vulnerabilities

Security vulnerabilities affecting SOA implemented using WS technology can be divided into four major groups (see Figure 2.6); Classical Vulnerabilities in hardware, operating systems and software used to build SOA middleware; Web Application (WA) vulnerabilities as SOA is commonly built on top of the Web protocols; and vulnerabilities due to the nature of SOA design, and new protocols and message formats supporting an SOA (the grey layers in Figure 2.6) [23]. All these security vulnerabilities are briefly discussed in the remainder of this section.

<table>
<thead>
<tr>
<th>Business Process Layer Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Services Layer Vulnerabilities</td>
</tr>
<tr>
<td>Web Application Vulnerabilities</td>
</tr>
<tr>
<td>Classical Vulnerabilities in Hardware, Operating Systems and Software</td>
</tr>
</tbody>
</table>

Figure 2.6: Security Vulnerabilities Affecting SOA (implemented using WSs technology) Layers [23]

Classical Security Vulnerabilities

Classical security vulnerabilities refer to security vulnerabilities in the existing operating systems, software and hardware infrastructure that could be exploited without the use of more recent web technologies (e.g., buffer overflow) [23]. These vulnerabilities are listed and updated in the U.S. NVD [25], OSVDB [26], US-CERT Vulnerability Notes Database [27], MITRE Common Vulnerabilities and Exposure [28] and SecurityFocus [29].

The existing operating systems, software and hardware infrastructure are employed to implement and deploy WSs hence, their related security vulnerabilities (listed in the above databases) also affect WSs.
Web Application Vulnerabilities

Web Application security vulnerabilities are related to the middleware and application layer of WSs. These vulnerabilities are classified in WASC [30], and a list of their most critical types is maintained by the OWASP [31].

WSs’ Specific Security Vulnerabilities

This type of security vulnerabilities is due to the WSs’ specific implementation technologies. Jensen et al. [9], [32] have presented a list of such security vulnerabilities, which are identified through exemplary attacks on widespread WS implementations. According to them, some of these vulnerabilities are due to implementation weaknesses, but the majority of them are due to protocol flaws. Similarly, Suriadi et al. [33] have investigated WSs’ specific security vulnerabilities to DoS attacks in well-known WS platforms including Java Metro, Apache Axis, and Microsoft .NET. The results from their experiments indicate that the majority of the WS platforms cope well with attacks targeting memory exhaustion. However, they are still vulnerable to attacks that target the CPU-time exhaustion. A number of WSs’ specific security vulnerabilities are as follows:

**WSDL Scanning**

As explained in Section 2.1, WSDL is an interface of a WS and advertises its operations, parameters, data types and network bindings. A WS may contain operations that should be accessed from the local network only, as well as operations that are intended to be offered to the outer network. In this case, separate WSDL, advertising the information about the external operations only, could be provided to the external clients to avoid these clients to get access to the internal operations. However, to invoke the external operations, the external clients should have access to the WS’s endpoint [32]. Therefore, an attacker can still try to guess the omitted operations and try to invoke them, which is called WSDL Scanning [32].
**Metadata Spoofing**

The metadata document of a WS contains all information, necessary for its invocation (e.g., message format, network location, and security requirements). This document is published by the WS's owner and is available to its clients. It is usually distributed using HTTP or mail communication protocols, which brings a Metadata Spoofing possibility. WSDL Spoofing and Security Policy Spoofing are two examples of attacks that exploit these vulnerabilities. For example, a WSDL Spoofing attacker may modify the network endpoint to perform man-in-the-middle attack for eavesdropping or modifying data [32].

**Attack Obfuscation**

As it will be introduced later, WS-Security [34] is a very flexible security standard that allows signing and encrypting only parts of the message, which contain sensitive data. However, an attacker may use an encrypted part to conceal malicious code for cyber-attacks such as Oversize Payload, Coercive Parsing, etc. [32]. Hence, the encrypted parts of the message should be inspected for the existence of such attacks. To enable examining the encrypted parts, they should first be decrypted, which is a disadvantage of using this standard. This type of attack may affect the availability of the WS in two ways [32]:

- **If the message is decrypted after the schema validation:** its malicious contents may pass the validation.
- **If the message is decrypted before schema validation:** the required XML and cryptographic processing may cause a long delay since decryption can be a performance-intensive process especially if the message contains malicious contents.

**Oversized Cryptography**

In WS-Security, there is no limit for the parts of the message that can be encrypted and for the size of the encrypted content. The flexibility of this standard allows a variety of security elements to be used in the WS-Security
header, which prevents strict schema validation and gives the possibility of the *Oversized Cryptography* attack [32]. This type of attack may affect the availability of the WS in three ways [32]:

- **If an oversized security header is used and the target system processes the entire security header**: the attack can have a similar effect on the targeted WS as an *Oversized Payload* attack.

- **If chained encrypted keys are used within the security header**: each encrypted key is used to encrypt the next key. Hence, the decryption process produces a very high CPU load.

- **If the SOAP message contains a large number of nested encrypted blocks**: a large number of cryptographic operations produce a very high CPU load.

**BPEL Scanning**

As it will be explained later, *BPEL* [35] is a de-facto standard and an executable business process modelling language enabling the modelling of the behaviour of a composite WS. Hence, it contains important information that can be used by an adversary to plot *BPEL Scanning* attacks (similar to *WSDL* scanning) on the business processes [23].

**BPEL State Deviation**

Many instances of a *BPEL* process may run concurrently, and their communication endpoints will be open for incoming messages at any time, which can be used by an attacker to plot *BPEL State Deviation* attack [32]. Two examples of such attack are as follows [32]:

- An attacker may flood a *BPEL* engine with messages that are correct in terms of their message structure but have no meaningful content (no correct instance identifier). These messages will eventually be discarded by the *BPEL* engine but after a significant amount of redundant work (reading and searching all existing process instances for a process matching the message). This redundant work can exhaust the computational resources of the *BPEL*. 
An attacker may use messages which contain correct instance identifiers but target a receive activity that is not enabled in the running business process instance.

**SOAPAction Spoofing**

As described in Section 2.1, a SOAP message package consists of a transport protocol header and an envelope (which itself consists of a header and a body). The first child element of the body contains the operation addressed by the SOAP request [32]. If HTTP transport protocol is used, an additional operation identifier element called SOAPAction can be added to the request’s header [32]. However, it gives the possibility of SOAPAction Spoofing attack in the following cases [32]:

- **When the requested operation is solely identified based on the SOAPAction value:** a man-in-the-middle attacker may try to invoke a different operation than the one specified in the SOAP body by adding the malicious operation to the SOAPAction header (HTTP header is not protected by WS-Security so it can be easily modified).

- **When the requested operation is solely identified based on the first child element of the SOAP body:** an attacker may bypass the HTTP gateway if it is configured only to accept the value added to the SOAPAction header. Then the WS’s logic will execute the operation in the first child element of the SOAP body regardless of the SOAPAction value.

**XML Injection**

An XML Injection attacker targets the integrity of the XML stream (e.g., SOAP message) by overwriting its static portions (e.g., by adding some contents containing XML tags) [32]. Using this method, an attacker may get access to restricted data.

**XML Denial of Service**

Early steps in processing a request SOAP message include parsing and transforming the contents of the message to be usable for the WS’s
backend applications. Therefore, an XML parser is an essential part of the WS’s application logic. SAX [36] and DOM [37] are two typical XML parsers.

DOM parsers read the whole XML stream into memory then create hierarchical objects for each node (an element, an attribute, etc.) that is referenced by the application logic. An attacker can plot a DoS attack on a DOM-based WS by inputting a large XML file [38]. Such attacks (e.g., Oversize Payload and Coercive Parsing) affect the availability of the WS by exhausting its resources and eliminating the legitimate user’s access [39].

On the other hand, SAX parsers perform XML parsing at the start or end of a node without loading the whole XML stream into memory (they load a maximum of two elements into memory at a time) [38]. Whenever the parser reaches a node, it triggers an event, and the program’s event handler starts processing the data [38].

StAX is another event-based XML parsing approach. However, instead of triggering an event, it waits for a method call to parse its corresponding operation [40].

DoS attacks are one of the most popular attacks, which can be performed through a variety of techniques. This type of attacks exploit the vulnerabilities in XML-based documents (e.g., SOAP messages) targeting the parsing mechanisms and other resources, affecting the availability of the WS. A large number of these attacks, targeting well-known companies such as VISA and PayPal suggests that they can be a serious threat to today’s IT infrastructure [41]. Some XML DoS attacks include:

- **HashDoS**: hash tables can be employed within a SOAP message to store values and their references (e.g., attributes and their corresponding namespace). Ideally, each key should represent a unique value. If different keys represent the same value, a collision will happen, which results in resource-intensive computation. An attacker can exploit the weak hash function to perform a DoS attack [42], [43].
- **Oversize Payload**: according to Meiko Jenson et al. [32], the majority of WS frameworks employ DOM-based XML processing models.
These parsers consume a memory much bigger than the size of the message (factor 2 to 30) [32]. Therefore, one easy way to perform DoS attack on WSs would be to query the service using a very large request message (*Oversize Payload*).

- **Coercive Parsing**: making XML parsing as complex as possible using a large number of namespace declarations, oversized prefix names/namespaces *URIs* or very deeply nested XML structures are other ways to perform DoS attack on WSs and is called *Coercive Parsing* [9], [32].
- **SOAP Array Attack**: an attacker may add a SOAP array (with a large number of elements) into the SOAP message, forcing the WS to reserve a large space in memory for these elements before parsing the message [44].
- **XML Attribute Count Attack**: XML does not limit the size of the contents (elements, attributes and attributes’ value) between the XML tags. Hence, an attacker may add a large number of attributes into the SOAP message and perform *XML Attribute Count* attack if no such limit is enforced by the developers of the WS [45].
- **XML Element Count Attack**: similarly to *XML Attribute Count* attack, an attacker may add a large number of non-nested elements into the SOAP message and perform *XML Element Count* attack if no limit is enforced to the size of the contents between the XML tags by the developers of the WS [45].

### 2.2.2 Existing Countermeasures and their Limitations

The previous subsection (2.2.1) introduced a number of security vulnerabilities, specific to WSs. Here, we present a number of existing countermeasures against these security vulnerabilities followed by a number of their limitations.

**Countermeasures**

Some of the existing countermeasures against previously introduced WSs’ specific security vulnerabilities are as follows:
WSs Security Standards

As described in Section 2.1, the communication between WSs is supported by XML-based protocols (e.g., SOAP messages) that are vulnerable to XML attacks [32]. As a result, organizations such as the OASIS and W3C consortia have developed various security standards, an overview of which is presented in Figure 2.7. The most commonly implemented security standards are described below [46]:

![Figure 2.7: Web services Security Standards – National Reference Model [46]](image)

- **WS-Security**: it describes how to use XML to sign, verify, encrypt and decrypt SOAP message exchanges [18]. It defines a SOAP extension within the security header of the SOAP message, enabling the association of the security tokens (UsernameToken, BinarySecureToken, and XML Tokens) with the message. In addition to message authentication, it provides message integrity and confidentiality through XML Signature and XML Encryption in conjunction with security tokens, respectively. **WS-Security** is very flexible and allows signing and encrypting only parts of the message that contain sensitive data.

- **WS-Security Policy**: the characteristic of the **WS-Security**, which allows parts of the XML document to be signed and encrypted, requires WS servers and clients to negotiate a security policy. This security policy allows defining the **WS-Security** elements that can be used (e.g., the algorithms and the required security tokens) and requires the
SOAP message to contain all the defined security tokens as a minimum. The WS-Security Policy standard provides the XML syntax for such security policy [47].

- **SAML**: it defines how authentication should be securely exchanged among the services [48].
- **XACML**: it is an XML-based technology for writing access control policies enabling developers to determine the resources that are allowed to be accessed by a user [49].
- **XML Encryption**: this standard provides data confidentiality by enabling the encryption of fragments of an XML. The encrypted fragment will be replaced with an EncryptedData element containing the ciphertext for the encrypted fragment. This standard also enables the creation of an EncryptedKey element, inside the security header, for hybrid encryption [50].
- **XML Signature**: this standard provides data integrity and authenticity by enabling the digital signature of fragments of an XML. The result of the signing operation will be added to the signature element of the security header [12].

**Schema Validation and Schema Hardening**

- **Schema Validation**: this approach is effective in defending against attacks that use messages not conforming to the WS’s description. It validates the incoming message against the XML schema generated from the WSDL of the WS [32].
- **Schema Hardening**: this approach is about restricting the WS’s description constructs to finite boundaries, removal of non-public operations from the WS description, etc. For example, Nils Gruschka and Norbert Luttenberger [51] proposed a system that generates XML schemas from the WSDL file of a WS, hardens it (e.g., converts the unbounded elements, maxOccurs="unbounded" to a finite number of elements, maxOccurs=" finite number"), then advertises the modified description to the WS’s clients. It also validates the incoming messages against this hardened description.
Possible usages of schema-hardening and schema validation to defend against cyber-attacks exploiting WSs’ specific security vulnerabilities are as follows:

- **Attack obfuscation**: validating the decrypted message [32].
- **XML Injection**: a strict schema validation on SOAP messages, including data type validation, to eliminate the possibility of an attacker getting access to restricted data [32].
- **Oversized Cryptography**: to only accept the security elements that are explicitly defined in the security policy (this approach is called Strict WS-SecurityPolicy Enforcement) [32]. However, more security tokens may be added to the SOAP message than those defined in the WS-Security Policy, making the system vulnerable to unbounded number of additional tokens that an attacker may add to a SOAP message, which may cause costly cryptographic computations. This side effect can be mitigated by putting a limit on the number of security tokens that the SOAP message may contain and rejecting messages exceeding this limit [32], [52].
- **Oversize Payload**: restricting the message size to a pre-defined limit and rejecting any larger messages. Currently, .NET 2.0 framework employs this method and rejects any message larger than 4MB, by default [32]. A more appropriate approach is to modify the XML schema used in the WS’s description then validate incoming SOAP messages against this schema [51].
- **Coercive Parsing**: using schema validation to defend against deeply nested XML [32].
- **SOAP Array**: enforcing strict schema validation on the maximum number of array elements. If it is not possible to put a limit on the number of array elements, comparing and validating the declared and existing number of elements in a SOAP array thereafter dropping the SOAP packet at the WS layer if the validation is failed [44].
- **XML Attribute Count and XML Element Count**: to put a limit on the size of the components (e.g., elements and attributes) within an XML
tag by the developers of the WS since the XML standard doesn’t impose such limit [45], [53].

- **SOAPAction Spoofing**: identifying the requested operation based on the first child element of the SOAP message. Additionally, the identified operation should be compared with the SOAPAction value and get rejected if it does not match [32].

- **Metadata Spoofing**: checking all metadata documents for authenticity [32].

**Firewall**

Sometimes an XML firewall is a more appropriate countermeasure against the attacks exploiting the WSs’ *XML-related* security vulnerabilities, such as:

- **WSDL Scanning**: all request messages (internal and external) have the same destination IP address, TCP port, and HTTP URL. Therefore, to reject invocation of the internal operation by the external clients, only a WS-aware XML firewall can distinguish whether an operation should be accessed by an external client [32].

- **BPEL State Deviation**: Gruschka et al. [54] proposed a firewall for processing and rejecting the invalid messages using as few computational resources as possible.

- **BPEL Scanning**: enforcing appropriate access control mechanisms on the internal operations [23].

**Other countermeasure approaches**

- Establishment of trust relationship prior to the communication (countermeasure against **Metadata Spoofing**) [32].

- Deploying internal and external operations on separate WSs, preferably on different servers (countermeasure against **WSDL Scanning** and **BPEL Scanning**) [32]
Limitations of the Introduced Countermeasures

Here, we present a number of the limitations related to the countermeasure approaches introduced previously:

- Equipping a SOAP message with WS-* security standards may require changes to the structure of the message since their use may introduce additional elements which did not exist previously.
- If the employed standard is revoked, the message structure may require significant changes.
- The difficulty, variety, and limitations of the WS-* standards as well as unfamiliarity of the developers with all of these standards have resulted in the development of WSs that are still vulnerable to cyber-attacks [18]. For example:
  - McIntosh and Austel [12] have shown that the content of a SOAP message, which is protected by an XML Signature (as specified in WS-Security) can be changed without invalidating the signature. The authors have shown that a signed element can be replaced with a fake element so that the signature remains valid.
  - The WS-Security standard is an important defence against the WSs’ specific cyber-attacks but only when its corresponding WS-SecurityPolicy is defined correctly. Otherwise, it may cause integrity and confidentiality vulnerabilities, for example, it may no longer require the SOAP message to contain all the security tokens as a minimum (see Section 2.2.2 for more details) [12], [55].
  - WS-SecurityPolicy header schema allows any kind and amount of security tokens and the encrypted blocks are allowed nearly everywhere within the SOAP message, which may cause costly cryptographic computations [32].
  - Improper use of SOAPAction standard opens a window to SOAPAction Spoofing attack.
Schema validation may cause high CPU load and large memory consumption [32].

In XML specifications, there is no limit specified for the number of namespace declarations per XML element and for the length of the namespace URIs (which enables Coercive Parsing attack). Arbitrary restrictions can be enforced on the number and length of the namespaces to defend WSs against the attacks misusing namespace declarations. However, it can result in unpredictable rejection of the messages [32].

Often validators rely on XSD derived from WSDL document or hand-coded by programmers, which may make them prone to cyber-attacks.

Similar to the previous limitation, manual unguided schema updates that rely entirely on the skills of a programmer may also make the validator prone to cyber-attacks.

If the schema used by the validator is loosely defined, it may allow malicious messages to pass through.

Schema restriction and validation cannot safeguard against some cyber-attacks exploiting XML-related security vulnerabilities. For example, Jenson et al. [13] have shown that even the most restrictive XML Schema validators may fail to defend against XML Signature Wrapping attacks.

The approaches explained above will be effective only when event-based message processing (e.g., SAX) is employed [32]. Otherwise, these protection systems themselves will be vulnerable to similar attacks [32].

Each countermeasure only defends against specific cyber-attacks.

2.2.3 Summary

This section presented the security issues related to the WSs and a number of the existing countermeasures against WSs’ specific security vulnerabilities (since the focus of the work presented in this dissertation is on the WSs’ XML-related security vulnerabilities). It then explained a number of limitations related to the presented approaches.
In addition to all the discussions presented in this section, the security issue gets more challenging when OTWSs are employed since they are ready-made black boxes of unknown quality and their security is out of the control of their clients. Hence, ITWS is a more appropriate approach to tolerating the security issues when OTWSs are used, which is about making the system to live with the security vulnerabilities of its constituent OTWSs to ensure delivery of sufficiently dependable service. This approach is the motivation for the work presented in this dissertation, and one of its key benefits is the possibility of ensuring correct behaviour in the presence of attacks. The next section gives a concise overview of the concepts and techniques of dependability and its relation to intrusion-tolerance.

2.3 Dependability and Intrusion-Tolerance

Dependability of a computing system is the capability to avoid failures that are more frequent or more severe, and outage durations that are longer than is acceptable to the user(s) [56]. It consists of threats to, the attributes of, and the means by which the dependability is attained (see Figure 2.8). Sections 2.3.1-2.3.4 present the dependability means.

![Dependability Tree](image)

**Figure 2.8:** The Dependability Tree [56]

2.3.1 Fault-Prevention

Faults may be prevented through utilization of quality control techniques during the process of design and manufacturing hardware and soft-
ware. These approaches may prevent operational physical faults, interaction faults and malicious faults [56].

### 2.3.2 Fault-Tolerance

Initially, practical techniques (e.g., error control code, diagnostics to locate failed components, etc.) were used to improve the reliability of the early computers [56]. At the same time Neumann, Moore, Shannon, and their successors developed theories of using redundant less reliable components to create reliable logics and structures [56]. Then Pierce unified these theories to the concept of failure tolerance and Avizienis [57] integrated redundancy with practical techniques (e.g., error detection and recovery) into the concept of Fault-Tolerance systems. Thereafter, Randell [58] introduced software Fault-Tolerance, which was later complemented by N-version programming [59].

The objective of Fault-Tolerance is to retain the delivery of correct service in the presence of active faults (faults that cause an error). It is generally implemented using error detection and subsequent system recovery approaches [56]:

- **Error detection**: it generates error signals or messages (within the system) and is divided into concurrent and preemptive error detection techniques, which operate at the run time and while service delivery is suspended, respectively.
- **Recovery**: it transforms the system state containing error(s) and/or fault(s) into a state without any discovered errors and faults. It consists of:
  - **Error handling**: eliminates errors from the system state and may have the following forms:
    - **Rollback**: returning the system back to a checkpoint state (the state saved prior to error detection).
    - **Compensation**: eliminating error through redundancies within an erroneous state.
- **Roll-Forward**: transforming the system state to a new state with no detected errors.

  - **Fault handling**: prevents determined faults from being activated again in four steps:
    - **Fault diagnosis**: identifying and recording the location and type of the cause(s) of error(s).
    - **Fault isolation**: excluding the faulty component(s), physically or logically, from further participation in service delivery.
    - **System reconfiguration**: switching to use spare component(s) or appointing the task(s) to fit component(s).
    - **System re-initialization**: checking, updating and recording the new configuration and updating system data and records.

The classes of faults that can be tolerated and the choice of *Fault-Tolerance* implementation are directly related to the underlying fault assumption [56]. However, redundancy is widely employed to implement *Fault-Tolerance* and is surveyed in the various literature [60]–[62]. Assuming that hardware components have an independent failure, channels with identical design may be used to tolerate operational physical faults. Whereas, to tolerate solid design faults, channels providing the same functionalities via separate designs and implementations (design diversity) should be used [56].

**Redundancy**

Antonio Carzaniga and his colleagues [63] describe redundancy as a system's capability of executing the same functionality in several execution environments or various ways (e.g., using different execution paths). Redundancy is believed to be a valid defence against physical faults. Running multiple replicas of the system and switching to the functioning one when a failure occurs is an example of using redundancy to overcome hardware faults [64]–[66]. Redundancy can also be applied to the code, data, and environment of a software system to overcome its non-physical faults (e.g., partial or
complete replication of the code, input data or execution environment, including the execution processes themselves) [67].

Redundancy from Architectural Viewpoint

From a software architecture point of view, redundancy can be divided into intra-components, which only changes the structure of a single component (e.g., wrappers that filter component interactions) and inter-components groups [63]. The Figures 2.9-2.11 present three different inter-component redundancy patterns: Sequential Alternative Pattern, Parallel Selection, and Parallel Evaluation.

Sequential Alternative Pattern

![Sequential Alternative Pattern Diagram](image)

Figure 2.9: Sequential Alternative Pattern [63]

In Sequential Alternative pattern (see Figure 2.9), an alternative program will be executed if the execution of the current program fails. In this design, a separate Adjudicator (e.g., a voting system) is connected to the end of each program to detect its failure and to validate its output. This pattern is employed in recovery blocks, service substitution approaches, etc. [63].

Parallel Selection Pattern
Parallel Selection pattern (Figure 2.10), consists of the concurrent execution of several programs. In this pattern, a separate Adjudicator (e.g., a voting system) is connected to the end of each program to detect its failure and to validate its output. This pattern is employed in self-checking programming [63].

**Parallel Evaluation Pattern**

*Parallel Evaluation* pattern (Figure 2.11), consists of the concurrent execution of several systems/programs and an Adjudicator (e.g., a voting system), which evaluates the result from those parallel executions to produce a correct result. This pattern is employed in *N-version programming* [63].

**N-version programming**

*N-version programming* is a technique that concurrently executes *N* different systems/programs, which provide the same functionality but are implemented differently and are fed with the same input configuration, then
forms a consensus on the output from all these systems/programs to produce the final output [68]. The purpose of this technique is to achieve fault tolerance, assuming that these different systems/programs will exhibit failure diversity [68].

**Diversity**

In 1975, Randell [58] introduced design diversity as a mechanism for software fault tolerance. He proposed using backup components (with an independent design from the main components) when main components are failed. Design diversity is now a recognised defence against design faults, and a comprehensive survey of its benefits is presented in [69].

Joseph and Avizienis [70] proposed the use of *diversity* as a means of improving security and discussed the feasibility of using diverse compilers (implemented using *N-version programming*) to detect and mask *Trojaned* compilers that infect the generated executables with viruses. Later, Forrest et al. [71], [72] and Littlewood and Strigini [73] argued the validity and effectiveness of using *diversity* to mitigate the effects of cyber-attacks. Classifications of *diversity* techniques for improving security are presented in [74], [75].

### 2.3.3 Fault-Removal

*Fault-Removal* may be performed during the development phase and operational life. The fault removal process during the development phase starts by checking whether the system adheres to given properties (verification phase). If it does not, the diagnosis phase starts checking for identification of the fault that is preventing the verification condition to be fulfilled. Following on identification of the fault, necessary corrections are performed (correction phase). Finally, the system goes through verification again to make sure that the correction has not caused any undesirable consequences. *Fault-Removal* during the operational life of a system can be divided into two groups of corrective maintenance and preventive maintenance. The corrective maintenance removes the faults after they have produced an error. Whereas, the preventive maintenance uncovers and removes the faults before they produce an error [56].
2.3.4 Fault-Forecasting

*Fault-Forecasting* is about the evaluation of the system behaviour (through modelling and testing, e.g., fault injection) with respect to fault occurrence or activation and has two aspects [56]:

- **Qualitative**, or ordinal, evaluation (e.g., failure mode and effect analysis): identification, classification, and ranking of the failure modes or the events that would lead to system failures.
- **Quantitative**, or probabilistic, evaluation (e.g., Markov chains and stochastic Petri nets): evaluation in terms of probabilities of the extent to which some of the attributes of dependability are satisfied.

2.3.5 Intrusion-Tolerance

*Fault-Tolerance* is not restricted to accidental faults [56]. Research on the integration of fault tolerance and the defences against deliberately malicious faults (e.g., security threats) was started in the mid-80’s [70], [76], [77] and since, designs have been proposed for the tolerance of intrusions, malicious logic and viruses [56]. Hence, dependability approaches may also be employed to implement *Intrusion-Tolerance*.

2.3.6 Summary

This section presented an overview of the concepts and techniques of dependability. It briefly introduced *Fault-Prevention*, *Fault-Tolerance* (including redundancy, *N-version programming*, and *diversity*), *Fault-Removal* and *Fault-Forecasting* concepts and explained that dependability approaches could also be employed to implement *Intrusion-Tolerance*.

As discussed previously, WSs allow companies and organizations to implement their core business only and outsource other service components (e.g., OTSWSs) over the Internet, either pre-selected or on-the-fly. WSs are at risk of security vulnerabilities related to their specific implementation technologies such as *XML* as well as those of their underlying platforms (e.g., operating systems and frameworks) and their applications (e.g., vulnerability
to *SQL Injection* attacks). Security issues become more challenging when *OTSWSs* are used since they are beyond the control of their clients. Hence, tolerating their security vulnerabilities through dependability techniques is a more appropriate approach.

In this work, the focus is on tolerating *XML*-related security vulnerabilities of *WSs* when *OTSWSs* are employed. It utilizes *WSs*’ composability, *PCA*, *CA* and penetration testing to implement an *ITWS* formed by *N-version programming* and *diversity*, providing *Fault-Tolerance* and *Fault-Prevention*. Also, the penetration test results of the *OTSWSs* give the providers of these services an opportunity to improve their security (*Fault-Removal*) in the future releases. The following sections briefly introduce each of these concepts and explain their role in this work.

### 2.4 Penetration Testing

Penetration testing is an attempt to break into a system not to exploit it, but rather to identify its weaknesses [78]. According to Tran and Dang [79], penetration testing is the simulation of attacks (that could be performed by real hackers) to identify security vulnerabilities of the target system. However, penetration testing is not a measure of true security, but it enables improving security by eliminating the anticipated security vulnerabilities. Despite advantages such as identifying security vulnerabilities hence providing countermeasures for them and assisting organizations to acknowledge the effectiveness or ineffectiveness of the implemented security measures, penetration testing may cause information disruption, denial of services, and information leakage, as the individuals performing penetration tests, are usually granted access to substantial amounts of sensitive information [80]. Hence, it might not be performed on the actual operational environment, which in turn may affect the test results. It also has other limitations, such as limitations of known exploit and experiment.

There are two types of penetration testing, black-box and white-box [78]. The black-box approach is commonly employed to test *WAs*. Regarding the *WSs* these tests are performed as follows:
Black-box penetration testing: the service under test is seen from the point of view of an attacker. The tester maliciously manipulates the SOAP messages being sent to the WS, then analyses the WSs’ response and its program execution.

White-box penetration testing: similar to static code analysis, this approach looks into the source code of the WS to identify its potential vulnerabilities.

Both black-box and white-box techniques can be used to assess the security of a WS [81]. However, the former approach enables the understanding of what an unknown attacker can achieve. It can be scripted and performed automatically by a penetration testing tool [18], [82]. Regarding the WSs’, automatic penetration testing tools can be divided into tools testing the security vulnerabilities specific (e.g., WS-Attacker [18]) and nonspecific (e.g., w3af [83]) to WSs. Otherwise, penetration testing can be done manually using other tools such as SoapUI [84].

In the work presented in this dissertation, penetration testing is used to identify the security vulnerabilities of the candidate OTSWSs, which enables the selection of the optimal set (set of OTSWSs with the most diverse security vulnerabilities) to implement the Intrusion-Tolerance. Diversity is particularly important because any common security vulnerabilities among OTSWSs (participating in ITWS) opens a window of opportunity for compromising all the OTSWSs, suffering from the targeted security vulnerability, at the same time. Hence, such effect should be diminished as much as possible.

2.5 Principal Component and Cluster Analysis

This section briefly introduces the PCA and CA and their role in the work presented in this dissertation.

2.5.1 Cluster Analysis

When dealing with large amounts of data, it is very important to be able to classify and group them according to various criteria. CA allows split-
ting a group of objects (e.g., data) into a number of homogeneous subgroups (or clusters) using various clustering algorithms [85]. Clustering is widely adopted in a variety of fields, ranging from engineering (e.g., machine learning and pattern recognition), computer sciences (e.g., web mining), medical sciences (e.g., genetics), to earth sciences (e.g., geography), social sciences (e.g., psychology), and economics (e.g., marketing) [86].

There is no universally agreed selection of features and clustering schemes [87]. Most researchers describe a cluster based on the internal homogeneity (within objects in each cluster) and the external separation (among the objects in different clusters) [88], [89]. Xu and Wunsch [86] have classified the clustering algorithms (each having various descriptions), some of which are presented below:

- **Hierarchical**: organizing data into a hierarchical structure according to the proximity matrix, which can be further divided to:
  - **Agglomerative**: starting with \( n \) clusters each containing only one object, followed by a series of merge operations until one cluster containing all the objects is left. Based on the different definitions for the distance between two clusters (e.g., single linkage, complete linkage), there are many agglomerative clustering algorithms.
  - **Divisive (e.g., MONA and DIANA [90])**: in contrast to agglomerative analysis, the divisive analysis starts with a cluster containing all the objects then follows a series of divisive operations, dividing objects among different clusters until only one object is left in each cluster.

- **Squared Error-Based (e.g., K-means [91])**: aims to partition \( n \) objects into \( k \) clusters so that each object belongs to the cluster with the nearest mean.

- **pdf Estimation via Mixture Densities (e.g., GMDD [92])**: from the probabilistic point of view, data objects are assumed to be generated according to several probability distributions. Hence, pdf Estimation
algorithm uses the probability distributions of the data objects to form clusters.

- **Graph Theory-Based (e.g., Chameleon [93]):** this algorithm uses graph theory to distribute objects among different clusters.

- **Combinatorial Search Techniques-Based:** aims to find the global or approximate optimum global for combinatorial optimization.

- **Fuzzy (e.g., FCM [94]):** in the fuzzy algorithm, there is no restriction for each object to only belong to one cluster, and an object can belong to all the clusters with certain membership degree.

- **Neural Networks-Based (e.g., SOFM [95]):** this algorithm uses neural networks to distribute objects among different clusters.

### 2.5.2 Principal Component Analysis

**PCA [96],** is a useful statistical technique for analysing datasets with high dimensions (when patterns are difficult to be found) by reducing their dimensions while retaining the variations among their data as much as possible [97]. Hence, it has found application in fields such as face recognition [98] and is a common technique for finding patterns in data of high dimensions [99]–[101].

Principal Components (PCs) are linear transformations of the original set of variables, which are uncorrelated and are ordered in a way that the first few PCs contain the most variation within the original dataset [97].
2.5.3 Application of PCA in Cluster Analysis

Sometimes (when the first few PCs contain cluster information) in literature, PCA is employed to reduce the dimensions of the dataset before CA, hoping to reduce the running time for CA’s computation using a fewer number of PCs [102]. However, the first few PC’s may not always contain cluster information [103]. Figures 2.12a and 2.12b illustrate two fictitious situations where the PCA pre-processing step before CA may and may not help, respectively. Figure 2.12a shows that the projection of the data points on the first PC (the diagonal line) clearly highlights the separation between the two clusters in the data. Whereas, in Figure 2.12b, the projection of the data points on the first PC (in the direction of x2) does not preserve the separation between the two clusters in the data. Therefore, there is a need to investigate the effectiveness of PCA as the pre-processing step to CA before adopting such approach.

2.5.4 Summary

This section briefly introduced the concepts of CA and PCA. It then explained the utilization of PCA, prior to CA, as a means of extracting the structure of the clusters through reduction of dimensions of the dataset and showed (Figure 2.12) that such approach may not always be useful and its effectiveness should be investigated before being adopted. Hence, for the purpose of this work first, the effectiveness of PCA as a pre-processing step...
to CA on results from penetration testing candidate OTSWSs is investigated. Then both PCA and CA along with further penetration testing are used to identify the optimal (most diverse in terms of their security vulnerabilities) set of OTSWSs for implementation of Intrusion-Tolerance.

2.6 WS Orchestration and Choreography

SOA enables the composition of several services with complementary functionalities to form a single value-added composite service and offer it to the clients through a single interface. Hence, a composite WS may consist of various other single and/or composite WSs.

The constituent services of a composite WS may become unreachable for various reasons (e.g., becoming the victim of a DoS attack), which may impact its dependability (e.g., its integrity as it may not be able to provide the requested service). Also, more suitable (in terms of their QoS) services, offering similar functionality as covered by composite WS’s constituent services, may become available. Hence, WSs capable of dynamic reconfiguration would be a practical solution to address such changes. In this way, composite WSs can react to the changes in a timely manner while using the most suitable available resources and avoiding long service disruptions due to off-line repairs. Therefore, an increasing number of today’s services are developed using dynamic composition of the available resources to address the clients’ complex demands.

Each building block of a composite service consists of its constitutive WSs (among which the tasks are divided) and a description of data, management and control flow between them [104]. Hence, a variety of specifications and standards has been introduced to support the implementation of composite WSs. This section reviews Web Service Choreography Interface (WSCI) [105], Business Process Modelling Language (BPML) [106], and BPEL [35].
2.6.1 WSCI

Sun, SAP, BEA, and Intalio developed the WSCI specification, which enables the description of the message to flow between the aggregated WSs in XML format. However, it does not support the definition of executable business processes. WSCI choreography is an extension to WSDL and describes the interactions between the operations described in the WSDL. It requires a WSCI interface for each WS participating in the interaction. It also supports conditional looping, parallel and sequential processing, and exception handling [107]. Figure 2.13 illustrates WSCI collaboration.

2.6.2 BPML

BPML is developed by Intalio, Sterling Commerce, SUN, and CSC. It has the underlying process execution as WSCI and similar process flow constructs and activities as BPEL. A developer can describe the public interactions between the WSs and develop private implementations using WSCI and BPML, respectively [107].

2.6.3 BPEL

Initially, Microsoft and IBM developed XLANG [108] and Web Services Flow Language [109], respectively, to support business flow design. Later Microsoft, IBM, Siebel Systems, and SAP combined these standards and
formed version 1.1 of the BPEL4WS, called BPEL. BPEL is a de-facto standard and an executable business process modelling language for implementing composite WSs. It is a workflow language used for composition, orchestration, and coordination of WSs. It provides an XML-grammar to model the behaviour of a composite WS, for example, sharing the tasks between the aggregated WSs and describing the control logic among them [107].

![BPEL4WS Process Flow](image)

Figure 2.14: BPEL4WS Process Flow [107]

BPEL is a layer on top of WSDL. It communicates with other WSs through their WSDL interfaces. In standard BPEL processes, the interactions with other WSs are modelled as PartnerLinks, which are defined statically. Each PartnerLink has a PartnerLinkType, indicating two WSDL PortTypes, one to be used by the external WS (partner) to communicate with the BPEL and the other to be used by the BPEL to communicate with the external WS. BPEL only abstractly refers to other WSs, and it is the responsibility of the execution engine to indicate which port (and therefore binding) should be used for each PortType [110]. In contrast with WSDLs, BPEL-based WSs are stateful and may have long-running interactions with other WSs. Hence, BPEL also supports the correlation of application data to the process instances.
BPEL supports modelling of executable\(^2\) and abstract\(^3\) business processes and offers basic (e.g., receive, reply, assign, invoke, etc.), structural (e.g., sequence, pick, while, forEach, etc.) and exceptions-handling activities [107]. It also supports synchronous and asynchronous processes [111]. Figure 2.14 presents a BPEL4WS process.

**BPEL Basic Activities**

A BPEL process starts with a receive activity, which accepts the service request. The reply activity returns the response from the BPEL process, in the request-response processes. BPEL’s invoke and assign activities, invoke a partner WS and copy data from one location within the BPEL process to another, respectively.

**BPEL Structural Activities**

BPEL’s structural activities consist of other BPEL activities (basic or structural) and define the business logic among them. BPEL supports sequential (through sequence activity) and parallel (e.g., forEach activity) processes, which are the aggregation of activities that will be executed in an ordered sequence and simultaneously, respectively:

- **pick activity**: enables the selection of one of the alternative BPEL paths [111].
- **if activity**: is a conditional construct for implementing a BPEL branch [111].
- **while and forEach activities**: provide loop constructs [111].
- **flow activity**: provides concurrent execution of BPEL activities [111].

**BPEL Variables**

BPEL’s variables are typed (WSDL message type, XML schema primitive type or XML schema element) and are used to store messages that are

---

\(^2\) Behaviour of the web services in a specific business process that can be executed by an orchestration engine

\(^3\) Message exchange between the participating web services
exchanged within the business process or to hold data that relates to the state of the process [111].

**BPEL Fault and Compensation Handling**

A *BPEL* process can be divided into hierarchically organized parts using *scope* activities, which provide behavioural context for other *BPEL* activities [111]. *Scope* activities allow the definition of faults, compensations, termination and event handlers for activities within their boundary. When a fault occurs within a business process (e.g., a fault is thrown by the *BPEL* process), the process may successfully complete its operation only if the fault is handled by a *scope* [111]. *BPEL* allows defining fault-handling activities (*catch* or *catchAll*) within a *faultHandler* construct. When a fault handler catches a fault, the execution of the activities within the *scope* (to which it is related) stops, and exception handling process begins. If no *catch* is selected and *catchAll* is not present, the fault will be re-thrown to the immediately enclosing *scope*, if present. Otherwise, the process will terminate abnormally [111].

In business processes, the compensation behaviour must be explicitly defined to reverse the effects of non-completed processes. A *compensationHandler* gathers all activities that have to be carried out to compensate the fault. If a *compensationHandler* is not defined for any given *scope*, the *BPEL* engine implicitly creates a default *compensationHandler*, which compensates all inner *scopes* [111].

**BPEL Extensibility**

*BPEL* is extensible and supports the inclusion of *Java* code snippets directly into the *BPEL* process. The benefits of this approach are speed and transactionality. However, the best practice is to incorporate only small segments of *Java* code (short utility-like operations rather than business code). Otherwise, the separation of the business logic from implementation will be lost [112].
BPEL Development Tool

There are different providers of BPEL engines (open source and commercial). Open source BPEL engines include ActiveBPEL [113], ApacheODE [114], Open ESB [115] and jBPM [116]. Commercial BPEL engines include Oracle BPEL Process Manager [117], SAP Exchange Infrastructure [118] and WebSphere Process Server [119].

2.6.4 Summary

This section briefly explained the composition of WSs and introduced three composition approaches (BPEL, WSCI, and BPML) among which BPEL and BPML support development of the executable business processes and WSCI provides a more choreographed approach [107]. As it is explained, BPEL enables concurrent invocation of WSs, which is used in this work to implement Intrusion-Tolerance (through N-version programming and diversity) in ApacheODE BPEL.

Also, OTSWSs may be updated (which may increase or decrease their security) or more secure OTSWSs (offering the same required functionality) may become available. Therefore, their clients should be able to replace them with more secure OTSWS, if they wish to. The replacement for a more secure OTSWS may need to be done dynamically as switching off the client’s system might not be acceptable. But, BPEL constructs only allow invocations of OTSWSs that either already have a statically defined PartnerLinks or have exactly the same interface matching an existing PartnerLink. Otherwise, to perform the replacement, the BPEL process has to be redeployed. However, dynamic invocations may still be achieved through Java snippet activities that are available as BPEL extensions. Hence, this work also demonstrates the reconfiguration of the ITWS using a combination of BPEL constructs and Java as BPEL extension approach as well as using only Java as BPEL extension approach, in Oracle BPEL.
2.7 Summary

This chapter first presented the overview of WSs’ architecture and introduced their main technologies (SOAP, WSDL, and UDDI). It then explained that WSs are at risk of security vulnerabilities related to their specific implementation technologies (e.g., XML) as well as those, of their underlying platforms (e.g., operating systems) and WA (e.g., vulnerability to SQL Injection attacks).

Afterward, it presented a number of existing countermeasures against attacks targeting XML-related vulnerabilities followed by a list of their limitations. It then argued that the issue gets more challenging when OTSWs are employed, as they are ready-made black boxes of unknown quality and their security is out of the control of their clients, thus, tolerating their security vulnerabilities through a reconfigurable ITWS is a more appropriate approach.

Then, it briefly introduced dependability approaches and explained that ITWS could be achieved using these techniques which (Fault-Tolerance and Fault-Prevention), in this work, are achieved through utilization of WSs’ composability, PCA, CA and penetration testing. Finally, it briefly introduced each of these concepts and explained the motivation for their adoption in this work. Next chapter presents the architecture of this reconfigurable ITWS.
The previous chapter briefly introduced Web Services (WSs) and their related security issues and discussed that a reconfigurable Intrusion-Tolerant Web Service (ITWS) is a suitable solution to their security issues, especially when Off-The-Shelf Web Services (OTSWs) are employed.

This chapter presents the architecture of the proposed self-reconfigurable ITWS, which can be deployed by the clients of OTSWs to diminish the impact of WSs’ (specifically XML-related) security vulnerabilities. It begins by explaining the overall objective (Section 3.1) followed by assumptions made (Section 3.2), the overview of the architecture (Section 3.3) and a summary of the discussions presented in this chapter (Section 3.4).

3.1 Objective

The objective of this architecture is to enable the transformation of an ordinary BPEL process that employs OTSW(s) into a self-reconfigurable ITWS as depicted in Figure 3.1. Through this architecture, every OTSW can be replaced by its equivalent self-reconfigurable ITWS (a group of OTSWs from various vendors, offering the desired functionality with security vulnerabilities diversity, that majority vote on their responses is considered as the final response).

---

4 If one group of OTSWs fails the business process switches to an alternative group
The steps to achieve the above objective are as follows:

1. Finding as many OTWSs (offering the desired functionality) as possible.
2. Penetration testing each of these services to identify their security vulnerabilities.
3. Performing statistical analysis on the penetration test results (from step 2) to form groups of 2f+1 services with security vulnerabilities diversity. This is because, with 2f+1 services, the majority of the responses remain correct even after as many as f failures (e.g., unavailability as a result of DoS attacks in the context of this work) [120], [121]. A system consisting of a number of distinct components is f-fault-tolerant if it satisfies its specification provided that no more than f of those components become faulty during some interval of interest.
The $f$-fault-tolerant is a measure of the Fault-Tolerance supported by the system architecture [120].

4. Penetration testing each group (from step 3) to identify the optimal set (the most secure among the groups) as well as to order each group in terms of their overall security vulnerabilities (ascending order starting with the most secure group).

5. Starting the business process with the first group (most secure group) and switch to the next group if the first group fails and so on.

6. Perform majority voting on the responses from OTSWSs within the running group to indicate the response of the ITWS.

### 3.2 Assumptions

This section presents the assumptions made in this framework:

- There are a sufficient number of candidate OTSWSs (offering the desired functionality) to implement Intrusion-Tolerance.
- Permission is granted by the owners of OTSWSs to perform penetration testing on these services since it may cause information disruption, denial of services and/or information leakage.
- It is possible to integrate the external penetration testing tool into this framework.

### 3.3 Architecture

![Figure 3.2: General Architecture, the dotted lines indicate the external systems](image)

Figure 3.2: General Architecture, the dotted lines indicate the external systems.
The architecture (see Figure 3.2) of this framework consists of various repositories, an Administrator, a Services Manager (SM), a Service-Groups Manager (SGM), a Business Process (BP) and two other business processes (one for testing OTWSs and one for testing groups of OTWSs). It is necessary to test OTWSs inside a business process as they may eventually participate in the BP. Hence, any effect from the business process engine should be taken into account when testing OTWSs. Also, separate business processes (with the exact setup as BP) than BP should be used for testing purposes to eliminate the effects of penetration testing on the operation of BP that executes the business logic.

Based on the case studies presented in Chapter 7, the dynamic adaptation using Java as BPEL extension only, and a combination of Java snippets and BPEL constructs can be implemented using about 400 and 260 lines of codes, respectively. The service selection is performed manually in the case studies (Chapters 4-6) aiming to evaluate the proposed service selection approach.

Repositories

This architecture consists of the following repositories:

- **Services Repository (Table 3.1):** is managed by the Administrator and is used by SM, SGM, and BP. It stores required information for invoking candidate OTWSs. Each OTWS may offer more than one service, which may be invoked at different stages of the BP, or have different versions. Hence, this repository stores each of these services separately with a unique Id.
  - **ID:** a unique Id is assigned to each service.
  - **Service Name:** stores the name of the OTWS.
  - **Endpoint:** stores the web address (URL) of the OTWS through which the operations provided by the OTWS are accessible.
  - **Target Namespace:** stores the namespace of the OTWS.
- **Invocation Id**: indicates at which stage of the *BP* this service may be invoked.
- **Operation Name**: each *OTSWS* may offer more than one services (operations). This column stores the name of the service that could be used in the *BP*.
- **Input**: stores the type and order of the service’s inputs parameters.
- **Output**: stores the type of the service’s output parameter.

- **Penetration Tests Repository (Table 3.2)**: is managed by the *Administrator* and is used by *SM* and *SGM*. It stores information about available penetration tests and their settings (the *Administrator* makes these decisions).
  - **Test Id**: each test may be performed with various settings (e.g., *Coercive Parsing* may be performed with various numbers of open tags). Hence, this repository stores each test separately with a unique *Id*.
  - **Test Name**: stores the name of the test.
  - **Test Setting**: stores the setting to be used for the test.

- **Services Penetration Tests Results Repository (Table 3.3)**: is managed by *SM* and is used by *SGM*. It stores the information about penetration test results of each *OTSWS*.
  - **Service Id**: stores the *Id* of the tested *OTSWS*.
  - **Test Id**: stores the *Id* of the penetration test performed on the *OTSWS*.
  - **Test Date**: stores the date when the penetration test is performed on the *OTSWS*.
  - **Test Result**: stores the penetration test result (e.g., whether the penetration test has been successful or not).

- **Service-Groups Repository (Table 3.4)**: is managed by *SGM* and is used by *BP*. Recall (see section 3.1) that in this framework *OTSWSs* are grouped based on their penetration test results and that the resultant groups are ordered in terms of their overall security vulnerabilities (ascending order starting with the most secure group), this repository stores information about service groups.
- **Service-Group Id**: it associates services to different service groups identified by a Service-Group Id.

- **Service Id**: stores the Id of the OTSWSs enabling BP to retrieve, from the SR, the required information for invoking them.

- **Invocation Id**: indicates at which stage of the BP the services from this group should be invoked.

- **Order Number**: indicates the order of the selection of the groups. The service group with the lowest orderNumber has the highest overall security among other groups, and the business process should start with that group and switch to the next group if it fails and so on.

- **Groups Penetration Tests Results Repository (Table 3.5)**: is managed and used by SGM. It stores the information about penetration test results of each group of OTSWS.

  - **Service-Group Id**: stores the Id of the tested service group.
  
  - **Test Id**: stores the Id of the penetration test performed on the service group.
  
  - **Test Date**: stores the date when the penetration test is performed on the service group.

  - **Test Result**: stores the penetration test result (e.g., whether the penetration test has been successful or not).

- **Failure Records Repository (Table 3.6)**: is managed by BP and is used by the Administrator. It stores information about runtime failures of each OTSWS.

**Table 3.1: Services Repository (SR)**

<table>
<thead>
<tr>
<th>ID</th>
<th>Service Name</th>
<th>Endpoint</th>
<th>Target Namespace</th>
<th>Invocation Id</th>
<th>Operation Name</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Service1</td>
<td>http://...</td>
<td>a.b.c</td>
<td>Inv1</td>
<td>Op1</td>
<td>type in₁, ... type inₙ</td>
<td>type out</td>
</tr>
<tr>
<td>S2</td>
<td>Service1</td>
<td>http://...</td>
<td>a.b.c</td>
<td>Inv6</td>
<td>Op2</td>
<td>type in₁, ... type inₙ</td>
<td>type out</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sn</td>
<td>Servicen</td>
<td>http://...</td>
<td>d.e.f</td>
<td>Inv1</td>
<td>Op1</td>
<td>type in₁, ... type inₙ</td>
<td>type out</td>
</tr>
</tbody>
</table>
Table 3.2: Penetration Tests Repository (PTR)

<table>
<thead>
<tr>
<th>Test Id</th>
<th>Test Name</th>
<th>Test Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Coercive Parsing</td>
<td>1500 open tags</td>
</tr>
<tr>
<td>T2</td>
<td>Coercive Parsing</td>
<td>2000 open tags</td>
</tr>
<tr>
<td>Tn</td>
<td>Hash Collision</td>
<td>1000 colliding attributes</td>
</tr>
</tbody>
</table>

Table 3.3: Services Penetration Tests Results Repository (SPTRR)

<table>
<thead>
<tr>
<th>Service Id</th>
<th>Test Id</th>
<th>Test Date</th>
<th>Test Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>T1</td>
<td>dd/mm/yyyy</td>
<td>Passed</td>
</tr>
<tr>
<td>S1</td>
<td>T2</td>
<td>dd/mm/yyyy</td>
<td>Failed</td>
</tr>
<tr>
<td>Sn</td>
<td>Tn</td>
<td>dd/mm/yyyy</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Table 3.4: Service-Groups Repository (SGR)

<table>
<thead>
<tr>
<th>Service-Group Id</th>
<th>Service Id</th>
<th>Invocation Id</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>S1</td>
<td>Inv1</td>
<td>1</td>
</tr>
<tr>
<td>G1</td>
<td>S2</td>
<td>Inv1</td>
<td>1</td>
</tr>
<tr>
<td>Gn</td>
<td>Sn</td>
<td>Inv1</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 3.5: Groups Penetration Tests Results Repository (GPTRR)

<table>
<thead>
<tr>
<th>Service-Group Id</th>
<th>Test Id</th>
<th>Test Date</th>
<th>Test Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>T1</td>
<td>dd/mm/yyyy</td>
<td>Passed</td>
</tr>
<tr>
<td>G1</td>
<td>T2</td>
<td>dd/mm/yyyy</td>
<td>Failed</td>
</tr>
<tr>
<td>Gn</td>
<td>Tn</td>
<td>dd/mm/yyyy</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Table 3.6: Failure Records Repository (FRR)

<table>
<thead>
<tr>
<th>Failure Id</th>
<th>Service Id</th>
<th>Failure Date</th>
<th>Failure Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>S1</td>
<td>dd/mm/yyyy</td>
<td>.. : ..</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fn</td>
<td>S1</td>
<td>dd/mm/yyyy</td>
<td>.. : ..</td>
</tr>
</tbody>
</table>

Administrator

An Administrator is a person (the only human in this architecture) who is in charge of:

- Instantiating the framework.
- Designing the BP.
- Finding desired OTSWSs and adding their records to the SR.
- Recording new versions of an existing OTSWS as a new service into SR (it facilitates N-version programming).
- Removing the record of OTSWSs from SR, FRR, and SPTRR, if they should no longer be used (are no longer available or their number of failures, reported by BP, is very high).
- Recording the information about available penetration tests along with their setting (a separate record for each setting) into the PTR.
- Removing the records for penetration tests (that are no longer available) from the PTR and SPTRR.

**Services Manager**

The role of the **SM** is to test the services and record the test results through the following steps:

1. Retrieves the information about the service from the **SR**.
2. Adds to the ‘Service Test Business Process’, the information required for invoking the service.
3. Retrieves the information about penetration tests and their settings from **PTR**.
4. Uses the ‘Service Test Business Process’, tests information collected in step 3, and the external penetration testing tool to test the service.
5. Records the test result into **SPTRR**.

![Figure 3.3: Combined SM and SGM Operation Intervals. The solid lines and dotted lines indicate SM and SGM operation intervals, respectively](image)

**SM** performs steps 1-5 above in the following cases:

- Periodically (refer to Figure 3.3, every i time units) tests all services recorded in **SR**. OTSWSs may go under un-versioned updates, which may affect their security vulnerabilities, for example, switching off the schema validation through WS’s source code: `((BindingProvider) wsClient).getRequestContext().put("set-jaxb-validation-event-handler", "false")`. Or services may be removed from **SR**. Hence, **SM** periodically tests all the services available in **SR** and updates **SPTRR**.
Periodically (refer to Figure 3.3, every $j$ time units) checks whether a new service or a new failure is recorded in $SR$ and $FRR$, respectively, and if it is, it tests that particular service. The runtime failure of an $OTSWS$, reported by $BP$, might be as a result of un-versioned changes to that service. Hence, it should be tested.

**Service-Groups Manager**

The role of the $SGM$ is to group and test services having the same $invocationId$. It fulfils this task through following steps:

1. Retrieves, from the $SPTRR$, the penetration tests results of all the $OTSWS$s having the same $invocationId$.
2. Performs statistical analysis and groups $OTSWS$s based on their security vulnerabilities diversity and record the result into $SGR$.
3. For every group:
   a. Retrieves from the $SR$ the record of every $OTSWS$ in that group.
   b. Adds the information required for invoking the group to the ‘Group Test Business Process’.
   c. Retrieves the information about penetration tests and their settings from $PTR$.
   d. Uses the ‘Group Test Business Process’, tests information collected in the previous step, and external penetration testing tool to test the group.
   e. Records the test result into $G PTRR$.
   f. Assigns an $orderNumber$ to each group after comparing their penetration test results.

$SGM$ repeats the above steps for every $invocationId$. Also, every $k$ time units (Refer to Figure 3.3) it checks $SPTRR$ for changes related to the addition of a new penetration test record or deletion of an existing one. In the event of these changes, $SGM$ repeats the above steps for all the services having the same $invocationId$ as the service that the change is related to. It then performs the grouping process from
scratch since addition or removal of an OTSWS requires the re-
creation of all related groups.

Business Process

BP uses SR and SGR to add groups of OTSWSs into the business
process. It starts with the groups having the lowest orderNumber (the most
secure among other groups). To catch any failure, BP invokes each group of
OTSWSs from inside a scope activity (introduced in the previous chapter)
and uses its faultHandler facility. In the event of group’s execution failure, BP
repeats their execution one more time, and if it fails again, BP records the
failure and switches to the group with the next orderNumber. BP adopts the
same approach throughout the business process execution cycle.

3.4 Summary

This chapter presented the architecture for the proposed framework
along with its objectives and the assumptions made. It then explained the
role of each of its components and the interactions among them. The next
chapter evaluates this framework through various case studies.
Previous chapters have presented the background and motivations, and the architecture for the proposed framework. As discussed previously, diversity is a valid and effective approach to mitigating the effects of cyber-attacks. In this work, diversity is achieved through penetration testing candidate OTSWSs. This chapter uses a case study to evaluate the feasibility of using penetration test results of candidate OTSWSs as a means of achieving diversity for implementation of ITWS.

In particular, Section 4.1 briefly introduces the third-party penetration testing tool that is used in the case studies presented in this work. Section 4.2 demonstrates the feasibility of ITWS; formed based on penetration tests results of candidate OTSWSs, in terms of mitigating the XML-related security vulnerabilities. Section 4.3 provides the summary of the discussions presented in this chapter.

4.1 Penetration Testing Tool, WS-Attacker

The proposed framework is not based on any specific penetration testing tool or method (the penetration testing tool itself is not part of the framework). Therefore, the type of penetration tests and how they should be performed depends on the selected penetration testing tool.

For evaluation of the proposed framework, a third-party penetration testing tool was required. WS-Attacker [18], SoapUI [84], WSFuzzer [122],
and WSFAggressor (which is also based on WS-Attacker) \cite{123} enable penetration testing WSs for XML-related security vulnerabilities, which is the focus of this work. However, except WS-Attacker, they either support fewer numbers of attack types and do not provide attack evaluation (in the case of SoapUI and WSFuzzer) or require access to the system under test (in the case of WSFAggressor) \cite{40}. Hence, WS-Attacker is selected, which also enables automatic penetration testing.

WS-Attacker is an open source penetration testing tool, which enables automatic tests for XML-specific security vulnerabilities. It consists of a framework and plugin architecture (see Figure 4.1).

![Figure 4.1: Overview of WS-Attacker and Its Processing Steps \cite{18}](image)

Figure 4.1 presents the component diagram for WS-Attacker, which shows that each plugin connects to its framework through an interface.

![Figure 4.2: Component Diagram for WS-Attacker](image)
4.1.1 WS-Attacker Framework

WS-Attacker’s framework is based on SoapUI [84] and sets up an environment for attacking WSs to identify their XML-related security vulnerabilities. It requires the user to load the Web Service Description Language (WSDL) document of the WS that should be tested. It then parses the loaded WSDL, presents the extracted operations to the user and generates Simple Object Access Protocol (SOAP) requests based on the extracted information.

To setup the WS-Attacker's environment, the user should:

1. Select the target operation
2. Send a test request to the target operation through the user interface provided by WS-Attacker. From the response of the WS to the test request, WS-Attacker can identify the normal state of the WS and analyse the attack results accordingly.
3. Select and configure the attack plugins related to the attacks against which the WS should be assessed.
4. Run the framework to perform the selected attacks on the WS.

Upon successful attack completion, WS-Attacker presents the attack result to the user. The attack result indicates whether the attack has been successful and its criticality score [18]. The attack's success (true or false) depends on whether a security vulnerability is detected. The score indicates the potential damage the attack can cause and is calculated by WS-Attacker in different ways for each attack plugin (for example, see SOAPAction Spoofing attack plugins in Section 4.1.2).

4.1.2 WS-Attacker Plugin

In WS-Attacker, the attacks are implemented as plugins. Each plugin is an implementation of a model of an adversary performing one type of attack and allows the user to set various parameters, such as the number of parallel attack threads, number of requests per thread, milliseconds between every test-probe request, and milliseconds between every attack request.
WS-Attacker is extendable and provides a plugin interface enabling new attack plugins to be added [18]. A number of attack plugins have been developed for WS-Attacker by its developers and other researchers [18], [40], [41].

**SOAPAction Spoofing Attack Plugins**

SOAPAction Spoofing is one of the attack plugins of WS-Attacker [18]. It provides automatic and manual attack options. In the automatic mode, it generates attack requests with all possible SOAPAction headers and sends them to the WS under test. However, in manual mode, the user is allowed to set the SOAPAction header manually. According to the developers of this attack plugin, possible responses from a WS to a SOAPAction Spoofing attack are as follows:

a) A SOAP fault message, which is the proper response to a SOAPAction Spoofing attack.

b) A server error or misconfiguration message.

c) Execution of the first operation in the body of the message, meaning that the SOAPAction header is only checked by the HTTP firewall while it is ignored by the WS’s logic.

d) Execution of the operation defined in the SOAPAction header, meaning that the operation to be executed is solely selected based on the SOAPAction value.

This plugin interprets (b), (c) or (d) responses (listed above) as a successful attack [18]. It also rates the vulnerability to the performed attack in percentage, based on the difficulty of the execution of the attack and the impact it may have on the WS under test. For example, response (d) indicates a more vulnerable WS because the attack is easier to execute, it only requires changes to the SOAPAction element [18].

Mainka et al. [18] have tested this plugin on Apache Axis2 v1.6.1, JBossWS Native 6.0, JBossWS CXF 7.0 and .Net WSs 3.0, WS frameworks and the results have shown the vulnerability of all these frameworks to this type of attack. Within the named frameworks, Apache Axis2 and .Net WSs
are identified as the most vulnerable frameworks, as they always execute the operations defined in the SOAPAction header.

**Denial of Service Attack Plugins**

Denial of Service (DoS) attacks are one of the most popular attacks, which can be performed through a variety of techniques [40]. History of such attacks, targeting well-known companies, such as VISA and PayPal, indicates that they can be a serious threat to today’s IT infrastructure [40]. Other examples of this type of attacks are called HashDoS attacks, which exploit the weakness of a hash-mapping algorithm implementation [42], [43]. Regarding the WSs, DoS attacks can exploit the vulnerabilities in XML-based documents (e.g., SOAP messages) targeting the parsing mechanisms and other resources, affecting the availability of the WS.

Falkenberg *et al.* [40] have developed a number of fully automatic DoS attack plugins for WS-Attacker enabling the identification of the WSs’ vulnerability to Coercive Parsing, SOAP Array, XML Attribute Count, XML Element Count, XML Entity Expansion, XML External Entity, XML Overlong Names and HashDoS attacks. In implementing these plugins, they have assumed that the tester does not have direct access to the WS under attack, and can only examine its vulnerability by sending payloads to its server then evaluating its response time. They have defined the response time as the time when the last byte of the request is sent to the server until the first byte of the response is received from the server. These attack plugins test a WS for vulnerability to DoS attacks as follows [40]:

1. They send the user-defined untampered requests (extended to the size of the tampered requests) to the WS then log the response times. According to the developers of these attack plugins, the extension to the size of the untampered requests does not cause DoS attack and if it does it will be clearly indicated in the result to eliminate false positive.
2. After a user-defined elapsed time, they send the tampered requests (with the same load patterns as the untampered ones) to the WS then log the response times.

3. They calculate the ratio of the median of the response times of the last 10 (or the maximum number of requests if less than 10 requests are sent) tampered requests to the median of the last 10 (or the maximum number of requests if less than 10 requests are sent) untampered requests (see Equation 4.1). The attack success is decided according to the threshold values (see Table 4.1) that are chosen based on pre-tests on vulnerable and non-vulnerable WSs.

\[
ARTR = \frac{\text{median response time of the last 10 tampered requests}}{\text{median response time of the last 10 untampered requests}}
\]  

(4.1)

**Table 4.1: WS-Attacker’s DoS Attack Success Metrics [40]**

<table>
<thead>
<tr>
<th>Metric Value</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTR &lt; 3</td>
<td>Payload ineffective</td>
</tr>
<tr>
<td>3 ≤ ARTR &lt; 6</td>
<td>Payload effective</td>
</tr>
<tr>
<td>6 ≤ ARTR</td>
<td>Payload highly effective</td>
</tr>
</tbody>
</table>

In designing these attack plugins, all major errors, such as the increase in response time because of various message sizes or various network loads, are eliminated [40].

These plugins also test for DoS attack’s effect on third-party users (who visit the WS under attack) by continuously sending test requests to the target WS at constant user-defined intervals and in parallel to testing the WS for vulnerability to DoS attack. The attack’s effect on the third-party users is decided by comparing the median of the response times of all simulated third-party requests (MRTTPR), after starting to send first tampered request, with the threshold values (see Table 4.2) that are considered an acceptable response time by the developers of these attack plugins. They believe that longer response times than these thresholds exponentially increases the users’ annoyance level [40].
Table 4.2: WS-Attacker’s DoS Attack’s Effect Metrics on Third-Party Users [40]

<table>
<thead>
<tr>
<th>Metric Value</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRTTTPR &lt; 2 sec</td>
<td>no or small effect on third-party users</td>
</tr>
<tr>
<td>2 ≤ MRTTTPR &lt; 5</td>
<td>third-party users are affected</td>
</tr>
<tr>
<td>5 ≤ MRTTTPR</td>
<td>third-party users are highly affected</td>
</tr>
</tbody>
</table>

Falkenberg et al. [40] have tested a number of these attack plugins on Apache Axis2, Apache CXF, Metro, and ASP .Net frameworks as well as the IBM DataPower XI50 XML Security Gateway. The results of these tests (see Table 4.3) show that ASP .Net and IBM XI50 are not vulnerable to any of these DoS attacks while Apache Axis2, Apache CXF, and Metro frameworks are vulnerable to at least one of these DoS attack.

Table 4.3: Results of Testing DoS Attack Plugins on a Number of Web service Frameworks [40]

<table>
<thead>
<tr>
<th>Attack Name</th>
<th>Axis2</th>
<th>CXF</th>
<th>Metro</th>
<th>ASP .Net</th>
<th>IBM XI50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coercive Parsing</td>
<td>Yes</td>
<td>Yes</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>DJBX31A Hash Collision</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>DJBX33A Hash Collision</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>DJBX33X Hash Collision</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>XML Attribute Count</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>XML Element Count</td>
<td>x</td>
<td>Yes</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>XML Entity Expansion</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>XML External Entity</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>XML Overlong Names</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

4.2 Case Study: Feasibility of ITWS Formed Based on Penetration Test Results of Candidate WSs

This section uses a case study (the author’s work published in [124]), to demonstrate the feasibility of using penetration test results of candidate OTSWSs as a means of achieving diversity for implementation of ITWS.

The objective of this case study is to exemplify the implementation of an ITWS using WSs selected based on their security vulnerabilities identified through penetration testing, and testing its effectiveness as a defence against XML-related DoS attacks. The remainder of this section presents the procedure of this case study.
4.2.1 WSs Preparation

For this case study the following stock purchase WSs is developed:

- Two WSs developed using Apache Axis2 1.5.1 framework and were deployed on Apache Tomcat 6.0.18 running on Intel® Core™ i5-3320M CPU @ 2.60GHz system with 7.88GB usable RAM and 64-bit Operating System.
- Two WSs developed using .NET 4.0 framework and were deployed on Intel® Core™ 2 Duo CPU P8400@ 2.26GHz system with 3.00GB RAM and 32-bit Operating System.

Also, one third-party ASP.NET WS [125], which provides similar stock purchase functionality, is employed.

4.2.2 Penetration Tests Settings

In this study, Coercive Parsing attack is performed on these services with WS-Attacker’s settings presented in Table 4.4.

Table 4.4: WS-Attacker’s Settings

<table>
<thead>
<tr>
<th>Test Message Settings</th>
<th>WS-Attacker’s other Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coercive Parsing with 75,000 open tags plotted in the body of the SOAP message</td>
<td>2 parallel attack threads, 4 requests per thread, 500 milliseconds between every testprobe request, 750 milliseconds between every attack request, 4 seconds server recovery time, 5 seconds stop after the last tampered request</td>
</tr>
</tbody>
</table>

4.2.3 WSs’ Penetration Test Results

Each WS is tested individually for security vulnerability to Coercive Parsing attack, using WS-Attacker’s settings shown in Table 4.4. The penetration test results of these WSs are presented in Table 4.5. In this table, 100% indicates that the WS is vulnerable to Coercive Parsing attack and 1% shows that it does not have this security vulnerability.
4.2.4 ITWS Implementation

The ITWS is implemented through the composition of the three ASP.NET services and one of the Axis2 services (four highlighted services in Table 4.5), using BPEL 2.0 plugin for Eclipse 3.4. It is then deployed on Apache Ode 1.2 runtime (on top of Apache Tomcat 6.0.18 server) on the same machine that is hosting the Axis2 WSs (Intel® Core™ i5-3320M CPU @ 2.60GHz system with 7.88GB usable RAM and 64-bit Operating System). Figure 4.3 presents the overview of this BPEL process.

![Figure 4.3: BPEL Diagram of the ITWS](image)

The operation of this ITWS is as follows:

1. Receives client’s request.
2. Initializes variable associated to each concurrent invocation process (each invocation process is responsible for invoking one of the ITWS’s four constituent services) to 0.
3. Passes the client’s request to each of the parallel invocation processes.
4. Each concurrent invocation process invokes its associated WS.
5. Each concurrent invocation process sets its associated variable to 1 upon receiving a response from its associated WS.
6. Once three responses are returned by three of the constituent services (three of the variables associated to the invocation processes are having the value of 1), the BPEL scope (the concurrent invocation processes are wrapped within a BPEL scope which provides exception handling, see Section 2.6 for further details) throws an exception.

7. The exception handler terminates the concurrent invocation processes.

8. The business process replies to the client.

4.2.5 ITWS’s Penetration Test Results

The ITWS is tested for security vulnerability to Coercive Parsing attack using WS-Attacker with the same settings that were used to test individual services (see Table 4.4).

<table>
<thead>
<tr>
<th>Penetration test result for Coercive Parsing attack</th>
<th>Axis2 WS</th>
<th>Axis2 WS</th>
<th>ASP.Net WS</th>
<th>ASP.Net WS</th>
<th>ASP.Net WS</th>
<th>ITWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>100%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
</tr>
</tbody>
</table>

The last column of Table 4.5 presents the penetration test result for this ITWS and indicates that it is not vulnerable to Coercive Parsing DoS attack (1% denotes that the WS under test does not have this security vulnerability).

4.3 Summary

This chapter briefly introduced the penetration testing tool that is used in the case studies presented in this work. It then demonstrated the feasibility of ITWS formed using WSs with security vulnerabilities diversity (identified through penetration testing) using a case study. Chapter 8 uses the outcome of this case study to evaluate the proposed framework.
In this work, *Intrusion-Tolerance* is implemented using Business Process Engineering Language (*BPEL*). Hence, the effect of *BPEL* on the security vulnerabilities of Off-The-Shelf Web Services (*OTSWSs*) should be considered in the selection of these services. This chapter uses a case study to demonstrate that *BPEL* could affect the security vulnerabilities of *WSs*. Hence, penetration testing for service selection should be performed while *OTSWSs* are wrapped in a *BPEL* process.

Sections 5.1 and 5.2 demonstrate the case study and the summary of the discussions presented in this chapter, respectively.

### 5.1 Case Study: Effects of BPEL on WSs’ XML-Related Security Vulnerabilities

This section uses a case study to demonstrate (from the point of view of *WS-Attacker*) that *BPEL* could affect the XML-related security vulnerabilities of *WSs*. The remainder of this section presents the procedure of this case study.

#### 5.1.1 WS Preparation

For this case study, a simple *WS* is developed using the source code presented in Code 5.1 and *Axis2 1.6.1* *WS* framework. This service is deployed on *Apache Tomcat 6.0.18* server running on *Intel® Core™ i5-3320M*
CPU @ 2.60GHz system with 7.88GB usable RAM and 64-bit Operating System. This WS provides two simple sum and factorial services (see Code 5.1).

Code 5.1: Source Code for the Developed WSs

```java
package Axis_Tom_6;
public class Axis_Tom_6_sum {
    public int addIntegers(int firstNum, int secondNum) {
        return firstNum + secondNum;
    }
    public int factorial(int n) {
        int result = 1;
        for (int i = 1; i <= n; i++) {
            result = result * i;
        }
        return result;
    }
}
```

5.1.2 WS Wrapped within BPEL process

![BPEL Process for Wrapping the Developed WSs](image)

Figure 5.1: BPEL Process for Wrapping the Developed WSs.
The service from Section 5.1.1 is wrapped in a BPEL process (see Figure 5.1) developed using BPEL 2.0 plugin for Eclipse Neon.2 4.6.2. And was deployed on Apache 1.3.4 runtime (on top of Tomcat 5.5.26 server) running on Intel® Core™ i5-3320M CPU @ 2.60GHz system with 7.88GB usable RAM and 64-bit Operating System. The operation of this BPEL process is as follows:

1. Receives client’s request.
2. Assigns the client’s request to the invocation process that invokes the WS (from Section 5.1.1).
3. Invokes the WS.
4. Assigns the response from the WS to the output of the BPEL process.
5. Replies the response from the WS to the client.

### 5.1.3 Penetration Tests Settings

In this study, Coercive Parsing, DJBX31A Hash Collision, and XML Attribute Count attacks are performed on these services (WSs from Sections 5.1.1 and 5.1.2). Table 5.1 presents the WS-Attacker’s settings for each of these tests.

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Test Message Settings</th>
<th>WS-Attacker’s other Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>Coercive Parsing attack with 3,000 open tags plotted in the header of the SOAP message</td>
<td>2 parallel attack threads, 4 requests per thread, 500 milliseconds between every testprobe request, 750 milliseconds between every attack request, 4 seconds server recovery time, 5 seconds stop after the last tampered request</td>
</tr>
<tr>
<td>Test 2</td>
<td>Coercive Parsing attack with 11,000 open tags plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 3</td>
<td>DJBX31A Hash Collision attack with 2,000 paired keys/values plotted in the header of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 4</td>
<td>XML Attribute Count attacks with 50,000 paired keys/values plotted in the header of the SOAP message</td>
<td></td>
</tr>
</tbody>
</table>

### 5.1.4 Attack Elements for Test SOAP Messages

In the penetration tests performed on these services, the attack element is either plotted in the header or the body of the test SOAP message as presented in Message 5.1 and Message 5.2, respectively.
Message 5.1: Test SOAP Message with Attack Element Plotted in its Header

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:axis="http://Axis_Tom_6">
  <soapenv:Header>
    <soapenv:Header>
    <soapenv:Body>
      <axis:addIntegers>
        <axis:firstNum>1</axis:firstNum>
        <axis:secondNum>2</axis:secondNum>
      </axis:addIntegers>
    </soapenv:Body>
  </soapenv:Envelope>
</soapenv:Envelope>
```

Message 5.2: Test SOAP Message with Attack Element Plotted in its Body.

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:axis="http://Axis_Tom_6">
  <soapenv:Header>
  <soapenv:Body>
    <attackElement>
      <axis:addIntegers>
        <axis:firstNum>1</axis:firstNum>
        <axis:secondNum>2</axis:secondNum>
      </axis:addIntegers>
    </attackElement>
  </soapenv:Body>
</soapenv:Envelope>
```

In each type of attack, the attack element is replaced as follows:

- **Coercive Parsing Attack**: `AttackElement` is replaced with various number of open tags (e.g., `<X><X>…</X></X>`).
- **DJBX31A Hash Collision Attack**: `AttackElement` is replaced with `<attackElement $$PAYLOADATTR$$> test </attackElement>` and `$PAYLOADATTR$` element is replaced with various number paired keys/values (e.g., `ttttt="0" ttttU="1" ttttv6="2"`).
- **XML Attribute Count Attack**: `AttackElement` is replaced with `<attackElement $$PAYLOADATTR$$> test </attackElement>` and
$\textbf{PAYLOADATTR}$ element is replaced with various number paired keys/values (e.g., a0="0" a1="1" a2="2").

5.1.5 Penetration Tests Results

The services developed in Sections 5.1.1 and 5.1.2 are tested using \textit{WS-Attacker}'s settings shown in Table 5.1. Figures 5.2 and 5.3 show the component diagram for direct penetration testing the \textit{Axis2 WS} (\textit{WS} from Section 5.1.1) and for penetration testing this \textit{WS} while it is wrapped in a \textit{BPEL} process (\textit{WS} from Section 5.1.2), respectively.

![Component Diagram for Direct Penetration Testing of the WS in this Case Study](image1)

Figure 5.2: The Component Diagram for Direct Penetration Testing of the WS in this Case Study

![Component Diagram for Penetration Testing of the WS in this Case Study while it is wrapped in a BPEL Process](image2)

Figure 5.3: The Component Diagram for Penetration Testing of the WS in this Case Study while it is wrapped in a BPEL Process.
Figures 5.4a-5.4d and 5.5a-5.5d show the penetration test results for each test attack performed on Axis2 WS (WS from Section 5.1.1) when it is tested directly and when it is tested while wrapped in a BPEL process (WS from Section 5.1.2), respectively. The penetration test results show the vulnerability of these WSs to the performed attacks in percentage so that 100% indicates that the WS is very vulnerable to the attack and 1% shows that it does not have this security vulnerability.

Each test is repeated ten times (to increase the confidence of the outcome of these tests) and the average of the results is considered. Because, these test plugins decide on the security vulnerability of the WSs based on their response times that slightly differs every time the same test is performed. The vertical and horizontal lines in these graphs (Figures 5.4a-5.4d and 5.5a-5.5d) show the result of the penetration test every time the same test is performed and the average of the penetration test results for each test, respectively.

These results demonstrate that BPEL affects the XM-related security vulnerabilities of WSs (from the point of view of WS-Attacker). In some cases (Coercive Parsing plotted in the header, DJBX31A Hash Collision, and XML Attribute Count attacks) it has improved the vulnerability, and in other cases, it has worsened it (Coercive Parsing plotted in the body attack). Hence, it is a better approach to perform penetration testing while OTWSs are wrapped in a BPEL process, as in this framework the ITWS is implemented using BPEL so its effects on the security vulnerabilities of the OTWSs should also be taken into account.
Figure 5.4: Results from Direct Penetration Testing Axis2-1.6.1 WS (for information about each test see Table 5.1)

Figure 5.5: Penetration Test Results for Axis2-1.6.1WS while it was wrapped in a BPEL Process (for information about each test see Table 5.1)
5.2 Summary

This chapter used a case study to demonstrate (from the point of view of WS-Attacker) that BPEL could affect the XML-related security vulnerabilities of WSs. It improved the security vulnerabilities to Coercive Parsing (plotted in the header), DJBX31A Hash Collision and XML Attribute Count attacks while increasing security vulnerability to Coercive Parsing (plotted in the body) attack. The outcome of this case study is used in the service selection case study, presented in the next chapter.
In this work, ITWS is achieved through penetration testing candidate OTSWSs, grouping these services based on their security vulnerabilities diversity (using PCA and CA), ordering the groups according to their overall security vulnerabilities (identified through penetration testing), and starting the ITWS with services from the most secure group and switch to the next group if it fails and so on.

This chapter uses a case study to explain and exemplify the proposed service selection framework. Sections 6.1 and 6.2 demonstrate the case study and the summary of the discussions presented in this chapter, respectively.

6.1 Case Study: Security-Aware Selection of Optimal Group of WSs using PCA and CA, for Implementation of ITWS

This section uses a case study to evaluate the effectiveness of PCA and CA utilization in security-aware service selection based on penetration tests results of the candidate OTSWSs. The objective of this case study is to explain and exemplify the proposed service selection framework. The remainder of this section presents the procedure of this case study.
6.1.1 WS Preparation

As previously discussed, this framework is based on diverse OTWSs (open source WSs). However, OTWSs could not be used for this case study because

1. There was not an adequate number of third-party WSs offering similar services (e.g., third-party WSs offering stock purchasing service).
2. We did not have permission to perform penetration testing on the available third-party WSs.

Hence, for this case study, six WSs are developed using the source code presented in Code 5.1 (Chapter 5) and various WSs’ frameworks. These services are deployed on either Apache Tomcat 6.0.18 or Apache Tomcat 7.0.72 servers running on Intel® Core™ i5-3320M CPU @ 2.60GHz system with 7.88GB usable RAM and 64-bit Operating System. However, in the utilization of this framework if more in-house WSs should be used (e.g., lack of adequate number of OTWSs), these WSs can be run on different servers (other than the one running the ITWS) to avoid scalability issues.

The details of the six services developed for this case study are as follows:

- S1: Axis1.4 deployed on Apache Tomcat 6.0.18 server
- S2: Axis2 1.5.1 deployed on Apache Tomcat 6.0.18 server
- S3: Axis2 1.6.1 deployed on Apache Tomcat 6.0.18 server
- S4: CXF 2.3.10 deployed on Apache Tomcat 6.0.18 server
- S5: CXF 2.5.11 deployed on Apache Tomcat 7.0.72 server
- S6: CXF 2.6.3 deployed on Apache Tomcat 7.0.72 server

The Web Service Description Language (WSDL) files of these services are presented in Appendices A-F.

6.1.2 WSs Wrapped within BPEL processes

Recall from the result of the case study presented in Chapter 5 that penetration testing should be performed on the services while they are
wrapped in a BPEL process (to also take into account the effects of the BPEL on the security vulnerabilities of the WSs), each of the WSs developed in the previous section are wrapped in the BPEL process as presented in Section 5.1.2 (see Chapter 5).

These business processes are developed using BPEL 2.0 plugin for Eclipse Neon.2 4.6.2 and are deployed on Apache 1.3.4 runtime (on top of Tomcat 5.5.26 server) running on Intel® Core™ i5-3320M CPU @ 2.60GHz system with 7.88GB usable RAM and 64-bit Operating System.

6.1.3 Penetration Tests Settings

Table 6.1: WS-Attacker’s Settings

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Test Message Settings</th>
<th>WS-Attacker’s other Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>Coercive Parsing attack with 8,000 open tags plotted in the header of the SOAP message</td>
<td>2 parallel attack threads, 4 requests per thread, 500 milliseconds between every testprobe request, 750 milliseconds between every attack request, 4 seconds server recovery time, 5 seconds stop after the last tampered request</td>
</tr>
<tr>
<td>Test 2</td>
<td>Coercive Parsing attack with 9,000 open tags plotted in the header of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 3</td>
<td>Coercive Parsing attack with 10,000 open tags plotted in the header of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 4</td>
<td>Coercive Parsing attack with 11,000 open tags plotted in the header of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 5</td>
<td>Coercive Parsing attack with 12,000 open tags plotted in the header of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 6</td>
<td>Coercive Parsing attack with 13,000 open tags plotted in the header of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 7</td>
<td>Coercive Parsing attack with 8,000 open tags plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 8</td>
<td>Coercive Parsing attack with 9,000 open tags plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 9</td>
<td>Coercive Parsing attack with 10,000 open tags plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 10</td>
<td>Coercive Parsing attack with 11,000 open tags plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 11</td>
<td>Coercive Parsing attack with 12,000 open tags plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 12</td>
<td>DJBX31A Hash Collision attack with 2,000 paired keys/values plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 13</td>
<td>DJBX31A Hash Collision attack with 3,000 paired keys/values plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 14</td>
<td>XML Attribute Count attack with 40,000 paired keys/values plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 15</td>
<td>XML Attribute Count attack with 50,000 paired keys/values plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 16</td>
<td>XML Attribute Count attack with 60,000 paired keys/values plotted in the body of the SOAP message</td>
<td></td>
</tr>
</tbody>
</table>
In this study, Coercive Parsing, DJBX31A Hash Collision, and XML Attribute Count attacks are performed on these services. Table 6.1 presents the WS-Attacker’s settings for each of these tests. In the remainder of this section, these tests are referenced by their *Id* as presented in Table 6.1.

### 6.1.4 Attack Elements for Test SOAP Messages

In the penetration tests performed on these services, the test SOAP messages and their attack elements were set as it is explained in Section 5.1.4.

### 6.1.5 Penetration Test Results of Candidate WSs

The services developed in Sections 6.1.2 are tested using WS-Attacker’s settings shown in Table 6.1. Figures 6.1-6.12 show the penetration test results for test attacks performed on these services (every two figures show the tests results related to one of these services). Due to the space limitation, the *Id* of the penetration tests are used in these graphs, for information about each test, see Table 6.1.

The penetration test results show the vulnerability of these WSs to the performed attacks in percentage so that 100% indicates that the WS is very vulnerable to the attack and 1% shows that it does not have this security vulnerability.

Each test is repeated ten times (to increase the confidence of the outcome of these tests) and the average of the results is considered. Because, these test plugins decide on the security vulnerability of the WSs based on their response times that slightly differs every time the same test is performed. The vertical and horizontal lines in these graphs (Figures 6.1-6.12) show the result of the penetration test every time the same test is performed and the average of the penetration test results for each test, respectively.
Figure 6.1: Penetration Tests Results for Tests 1-8 Performed on Axis1-4 WS (for information about each test and WS see Table 6.1 and Section 6.1.1, respectively)
Figure 6.2a: Results from Test 9

Figure 6.2b: Results from Test 10

Figure 6.2c: Results from Test 11

Figure 6.2d: Results from Test 12

Figure 6.2e: Results from Test 13

Figure 6.2f: Results from Test 14

Figure 6.2g: Results from Test 15

Figure 6.2h: Results from Test 16

Figure 6.2: Penetration Tests Results for Tests 9-16 Performed on Axis1-4 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively
Figure 6.3: Penetration Tests Results for Tests 1-8 Performed on Axis2-1.5.1 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively.

Figure 6.3a: Results from Test 1

Figure 6.3b: Results from Test 2

Figure 6.3c: Results from Test 3

Figure 6.3d: Results from Test 4

Figure 6.3e: Results from Test 5

Figure 6.3f: Results from Test 6

Figure 6.3g: Results from Test 7

Figure 6.3h: Results from Test 8
Figure 6.4: Penetration Tests Results for Tests 9-16 Performed on Axis2-1.5.1 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively
Figure 6.5: Penetration Tests Results for Tests 1-8 Performed on Axis2-1.6.1 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively
Figure 6.6: Penetration Tests Results for Tests 9-16 Performed on Axis2-1.6.1 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively
Figure 6.7: Penetration Tests Results for Tests 1-8 Performed on CXF-2.3.10 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively.
Figure 6.8: Penetration Tests Results for Tests 9-16 Performed on CXF-2.3.10 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively
Figure 6.9: Penetration Tests Results for Tests 1-8 Performed on CXF-2.5.11 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively.
Figure 6.10: Penetration Tests Results for Tests 9-16 Performed on CXF-2.5.11 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively.
Figure 6.11: Penetration Tests Results for Tests 1-8 Performed on CXF-2.6.3 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively.
Figure 6.12: Penetration Tests Results for Tests 9-16 Performed on CXF-2.6.3 WS for information about each test and WS see Table 6.1 and Section 6.1.1, respectively.
6.1.6 Principal Components Analysis of Candidate WS’s Penetration Test Results

The objective of the service selection framework in this dissertation is to select group of OTSWSs with security vulnerabilities diversity (identified through penetration testing) for implementation of ITWS. To facilitate this service selection approach, OTSWSs should be divided into a number of groups according to their security vulnerabilities similarity, so that ITWS can be formed using services selected from these diverse groups.

Division of the OTSWSs can be achieved through CA. However, most CA algorithms cannot deal with data with high dimensionality (e.g., in the case of the proposed service selection framework, more than one OTSWSs each with various penetration test results) [86]. Hence, dimensionality reduction is important in CA, which reduces the computational costs and provides users with visual examination of the data. However, it causes loss of information. One practical approach for dimensionality reduction is to extract important components from the original data, which can contribute to forming the clusters [86].

PCA is one of the typical dimensionality reduction approaches, which is mainly used to reduce the dimensionality of the dataset while retaining the maximum information. Also, it is a common technique for finding patterns in data of high dimensions. Consequently, Principal Components (PCs) may be used as inputs for CA.

This section explains the steps for finding PCs of the WSs’ penetration test results, obtained in Section 6.1.5. It also demonstrates the effectiveness of PCA in reducing the computational complexity of CA in the proposed service selection approach.

**Step1: Collecting the Dataset**

The first step in PCA is about collecting the dataset (e.g., observations in an experiment). In this work, the input to the PCA is the penetration test results of the WSs from Section 6.1.5, which has 16x6 dimensions (six-
teen penetration test results for each of the six WSs). Table 6.2 and Figure 6.13 show the tabular and graphical representations of this data, respectively.

![Figure 6.13: Plot of Candidate WSs' Penetration Tests Results, to be used as PCA inputs (for information about each test and WS see Table 6.1 and Section 6.1.1, respectively)](image)

**Step 2: Subtracting the Mean**

For PCA to work properly, the mean (the average across each data dimension) should be subtracted from each of the data dimensions to ensure that the first PC describes the direction of maximum variance, which eliminates misleading directions. The mean matrix (average of each test’s results) and mean adjusted penetration test results are presented in Tables 6.3 and 6.4, respectively.

**Step 3: Calculating the Covariance Matrix**

PCA uses correlation/covariance matrices of the original variables to find new directions. In literature, a number of authors prefer to use correlation matrix to find principal components [102]. However, the general rule of thumb is to use correlation matrix if the variables are of different units (scale) as it standardises the data (see [97] for further information and examples). Since in this case study the penetration test results are of the same unit, the covariance matrix (presented in Table 6.5) is used for PCA calculation, and it is worked out using mean adjusted penetration test results (from Table 6.4) and MATLAB’s cov() command.
Table 6.2: WSs’ Penetration Test Results to be used as PCA input (for information about each test and WS see Table 6.1 and Section 6.1.1, respectively)

<table>
<thead>
<tr>
<th></th>
<th>Test1</th>
<th>Test2</th>
<th>Test3</th>
<th>Test4</th>
<th>Test5</th>
<th>Test6</th>
<th>Test7</th>
<th>Test8</th>
<th>Test9</th>
<th>Test10</th>
<th>Test11</th>
<th>Test12</th>
<th>Test13</th>
<th>Test14</th>
<th>Test15</th>
<th>Test16</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>73.8</td>
<td>100</td>
<td>1</td>
<td>56</td>
<td>75.5</td>
<td>98.6</td>
<td>100</td>
<td>66.5</td>
<td>100</td>
<td>64.7</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>55.5</td>
<td>66.7</td>
<td>87.4</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>59.2</td>
<td>75</td>
<td>97.2</td>
<td>100</td>
<td>1</td>
<td>98.2</td>
<td>87.7</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>58.1</td>
<td>67.9</td>
<td>87.2</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>58.4</td>
<td>75.5</td>
<td>97.7</td>
<td>100</td>
<td>1</td>
<td>98.5</td>
<td>86.8</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>69.3</td>
<td>96.2</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>62.2</td>
<td>89.6</td>
<td>100</td>
<td>1</td>
<td>64.5</td>
<td>1</td>
<td>71.7</td>
<td>100</td>
</tr>
<tr>
<td>S5</td>
<td>1</td>
<td>1</td>
<td>55.3</td>
<td>69.1</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>62.3</td>
<td>89.5</td>
<td>100</td>
<td>85</td>
<td>1</td>
<td>74.1</td>
<td>1</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>S6</td>
<td>1</td>
<td>1</td>
<td>54.7</td>
<td>68.1</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>62.9</td>
<td>89.7</td>
<td>100</td>
<td>84.3</td>
<td>1</td>
<td>74.7</td>
<td>1</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 6.3: Mean Matrix of the Penetration Test Results

<table>
<thead>
<tr>
<th>Average of Test results</th>
<th>Average of Test2 results</th>
<th>Average of Test3 results</th>
<th>Average of Test4 results</th>
<th>Average of Test5 results</th>
<th>Average of Test6 results</th>
<th>Average of Test7 results</th>
<th>Average of Test8 results</th>
<th>Average of Test9 results</th>
<th>Average of Test10 results</th>
<th>Average of Test11 results</th>
<th>Average of Test12 results</th>
<th>Average of Test13 results</th>
<th>Average of Test14 results</th>
<th>Average of Test15 results</th>
<th>Average of Test16 results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000</td>
<td>19.6000</td>
<td>41.4333</td>
<td>63.6833</td>
<td>95.0000</td>
<td>100.00</td>
<td>1.0000</td>
<td>29.4333</td>
<td>68.9000</td>
<td>93.7167</td>
<td>100.00</td>
<td>50.3833</td>
<td>99.4500</td>
<td>40.3667</td>
<td>86.7500</td>
<td>100.0000</td>
</tr>
<tr>
<td>1.0000</td>
<td>19.6000</td>
<td>41.4333</td>
<td>63.6833</td>
<td>95.0000</td>
<td>100.00</td>
<td>1.0000</td>
<td>29.4333</td>
<td>68.9000</td>
<td>93.7167</td>
<td>100.00</td>
<td>50.3833</td>
<td>99.4500</td>
<td>40.3667</td>
<td>86.7500</td>
<td>100.0000</td>
</tr>
<tr>
<td>1.0000</td>
<td>19.6000</td>
<td>41.4333</td>
<td>63.6833</td>
<td>95.0000</td>
<td>100.00</td>
<td>1.0000</td>
<td>29.4333</td>
<td>68.9000</td>
<td>93.7167</td>
<td>100.00</td>
<td>50.3833</td>
<td>99.4500</td>
<td>40.3667</td>
<td>86.7500</td>
<td>100.0000</td>
</tr>
<tr>
<td>1.0000</td>
<td>19.6000</td>
<td>41.4333</td>
<td>63.6833</td>
<td>95.0000</td>
<td>100.00</td>
<td>1.0000</td>
<td>29.4333</td>
<td>68.9000</td>
<td>93.7167</td>
<td>100.00</td>
<td>50.3833</td>
<td>99.4500</td>
<td>40.3667</td>
<td>86.7500</td>
<td>100.0000</td>
</tr>
<tr>
<td>1.0000</td>
<td>19.6000</td>
<td>41.4333</td>
<td>63.6833</td>
<td>95.0000</td>
<td>100.00</td>
<td>1.0000</td>
<td>29.4333</td>
<td>68.9000</td>
<td>93.7167</td>
<td>100.00</td>
<td>50.3833</td>
<td>99.4500</td>
<td>40.3667</td>
<td>86.7500</td>
<td>100.0000</td>
</tr>
</tbody>
</table>

Table 6.4: Mean Adjusted Penetration Test Results

<table>
<thead>
<tr>
<th>Test1 - Average of Test results</th>
<th>Test2 - Average of Test results</th>
<th>Test3 - Average of Test results</th>
<th>Test4 - Average of Test results</th>
<th>Test5 - Average of Test results</th>
<th>Test6 - Average of Test results</th>
<th>Test7 - Average of Test results</th>
<th>Test8 - Average of Test results</th>
<th>Test9 - Average of Test results</th>
<th>Test10 - Average of Test results</th>
<th>Test11 - Average of Test results</th>
<th>Test12 - Average of Test results</th>
<th>Test13 - Average of Test results</th>
<th>Test14 - Average of Test results</th>
<th>Test15 - Average of Test results</th>
<th>Test16 - Average of Test results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>-18.6</td>
<td>-40.4333</td>
<td>-62.6833</td>
<td>-21.2</td>
<td>0</td>
<td>0</td>
<td>26.5667</td>
<td>6.6</td>
<td>4.8833</td>
<td>0</td>
<td>16.1167</td>
<td>0.55</td>
<td>24.3333</td>
<td>13.25</td>
<td>0</td>
</tr>
<tr>
<td>0.00</td>
<td>35.9</td>
<td>27.2667</td>
<td>23.7167</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>29.7667</td>
<td>6.1</td>
<td>3.4833</td>
<td>0</td>
<td>-49.3833</td>
<td>-1.25</td>
<td>-47.3333</td>
<td>13.25</td>
<td>0</td>
</tr>
<tr>
<td>0.00</td>
<td>38.5</td>
<td>26.4667</td>
<td>23.5167</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>28.9667</td>
<td>6.6</td>
<td>3.9833</td>
<td>0</td>
<td>-49.3833</td>
<td>-0.95</td>
<td>46.4333</td>
<td>13.25</td>
<td>0</td>
</tr>
<tr>
<td>0.00</td>
<td>-18.6</td>
<td>-40.4333</td>
<td>5.6167</td>
<td>1.2</td>
<td>0</td>
<td>0</td>
<td>-28.4333</td>
<td>-6.7</td>
<td>-4.1167</td>
<td>0</td>
<td>14.1167</td>
<td>0.55</td>
<td>-39.3667</td>
<td>-15.05</td>
<td>0</td>
</tr>
<tr>
<td>0.00</td>
<td>-18.6</td>
<td>13.8667</td>
<td>5.4167</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>-28.4333</td>
<td>-6.6</td>
<td>-4.2167</td>
<td>0</td>
<td>34.6167</td>
<td>0.55</td>
<td>-39.3667</td>
<td>-12.95</td>
<td>0</td>
</tr>
<tr>
<td>0.00</td>
<td>-18.6</td>
<td>13.2667</td>
<td>4.4167</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>-28.4333</td>
<td>-6</td>
<td>-4.0167</td>
<td>0</td>
<td>33.9167</td>
<td>0.55</td>
<td>-39.3667</td>
<td>-12.05</td>
<td>0</td>
</tr>
</tbody>
</table>
### Table 6.5: Covariance Matrix of the Mean Adjusted Penetration Test Results

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.031</td>
<td>0.5995</td>
<td>0.5271</td>
<td>0.1156</td>
<td>0</td>
<td>0</td>
<td>0.6553</td>
<td>0.1419</td>
<td>0.0835</td>
<td>0</td>
<td>-1.1022</td>
<td>-0.0245</td>
</tr>
<tr>
<td>0.031</td>
<td>0.5995</td>
<td>1.0164</td>
<td>0.742</td>
<td>0.2426</td>
<td>0</td>
<td>-0.1755</td>
<td>0.0348</td>
<td>0.0151</td>
<td>-0.5892</td>
<td>-0.0178</td>
<td>0.4119</td>
<td>0.0899</td>
</tr>
<tr>
<td>0.5271</td>
<td>0.742</td>
<td>1.025</td>
<td>0.3242</td>
<td>0.1041</td>
<td>0</td>
<td>-0.1435</td>
<td>-0.0427</td>
<td>-0.0387</td>
<td>-0.5852</td>
<td>-0.0156</td>
<td>0.0162</td>
<td>-0.0822</td>
</tr>
<tr>
<td>0.1156</td>
<td>0.2426</td>
<td>0.3242</td>
<td>0.1102</td>
<td>0.1041</td>
<td>0</td>
<td>-0.1176</td>
<td>0.2925</td>
<td>-0.0225</td>
<td>-0.0952</td>
<td>-0.0033</td>
<td>-0.0976</td>
<td>-0.058</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.6553</td>
<td>0.1765</td>
<td>-0.1435</td>
<td>-0.1176</td>
<td>0</td>
<td>0</td>
<td>0.9713</td>
<td>0.2194</td>
<td>0.14</td>
<td>-0.9645</td>
<td>-0.0194</td>
<td>1.3517</td>
<td>0.4521</td>
</tr>
<tr>
<td>0.1419</td>
<td>0.0348</td>
<td>-0.0427</td>
<td>-0.0295</td>
<td>0</td>
<td>0</td>
<td>0.2194</td>
<td>0.0498</td>
<td>0.0319</td>
<td>-0.2095</td>
<td>-0.0042</td>
<td>0.3031</td>
<td>0.1025</td>
</tr>
<tr>
<td>0</td>
<td>0.0835</td>
<td>0.0115</td>
<td>-0.0387</td>
<td>-0.0225</td>
<td>0</td>
<td>0.14</td>
<td>0.0319</td>
<td>0.0205</td>
<td>-0.1261</td>
<td>-0.0024</td>
<td>0.191</td>
<td>0.0655</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1.1022</td>
<td>-0.5892</td>
<td>-0.5852</td>
<td>-0.0952</td>
<td>0</td>
<td>-0.9645</td>
<td>-0.2095</td>
<td>-0.1261</td>
<td>1.537</td>
<td>0.0326</td>
<td>-1.4984</td>
<td>-0.4308</td>
</tr>
<tr>
<td>0</td>
<td>-0.0245</td>
<td>-0.0178</td>
<td>-0.0156</td>
<td>-0.0033</td>
<td>0</td>
<td>-0.0194</td>
<td>-0.0042</td>
<td>-0.0024</td>
<td>0.0326</td>
<td>0.0007</td>
<td>-0.031</td>
<td>-0.0087</td>
</tr>
<tr>
<td>0</td>
<td>1.0462</td>
<td>0.4119</td>
<td>0.0162</td>
<td>-0.0976</td>
<td>0</td>
<td>1.3517</td>
<td>0.3031</td>
<td>0.191</td>
<td>-1.4984</td>
<td>-0.031</td>
<td>1.9276</td>
<td>0.6259</td>
</tr>
<tr>
<td>0</td>
<td>0.2957</td>
<td>0.0899</td>
<td>-0.0822</td>
<td>-0.058</td>
<td>0</td>
<td>0.4521</td>
<td>0.1025</td>
<td>0.0655</td>
<td>-0.4308</td>
<td>-0.0087</td>
<td>0.6259</td>
<td>0.2117</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 6.6: Eigenvectors for the Covariance Matrix of the Mean Adjusted Penetration Test Results

<p>| | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0339</td>
<td>0.0174</td>
<td>-0.0362</td>
<td>0.1167</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0006</td>
<td>-0.1694</td>
<td>-0.0402</td>
<td>-0.2966</td>
<td>-0.8257</td>
<td>-0.0988</td>
</tr>
<tr>
<td>-0.1008</td>
<td>-0.0308</td>
<td>-0.1167</td>
<td>0.0391</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0001</td>
<td>-0.0452</td>
<td>-0.2682</td>
<td>-0.5186</td>
<td>0.0608</td>
<td>0.7843</td>
</tr>
<tr>
<td>0.2816</td>
<td>0.1795</td>
<td>0.0716</td>
<td>-0.1005</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0005</td>
<td>-0.2117</td>
<td>0.2111</td>
<td>-0.3903</td>
<td>-0.289</td>
<td>-0.3381</td>
</tr>
<tr>
<td>-0.1041</td>
<td>-0.1006</td>
<td>0.6527</td>
<td>-0.2149</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0003</td>
<td>-0.6089</td>
<td>0.1838</td>
<td>0.2198</td>
<td>0.0305</td>
<td>-0.04</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.9855</td>
<td>0.1696</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.1696</td>
<td>-0.9855</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.1848</td>
<td>0.4519</td>
<td>0.3634</td>
<td>-0.0884</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0003</td>
<td>-0.0378</td>
<td>-0.4979</td>
<td>-0.2606</td>
<td>0.2091</td>
</tr>
<tr>
<td>-0.2999</td>
<td>0.5635</td>
<td>0.2689</td>
<td>0.1189</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0007</td>
<td>0.4139</td>
<td>0.3973</td>
<td>0.3895</td>
<td>-0.2186</td>
</tr>
<tr>
<td>-0.085</td>
<td>-0.0802</td>
<td>-0.0784</td>
<td>-0.9334</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0005</td>
<td>-0.2485</td>
<td>-0.0401</td>
<td>0.0497</td>
<td>-0.1915</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-0.0004</td>
<td>0.0011</td>
<td>0.0052</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.1904</td>
<td>0.1146</td>
<td>0.1894</td>
<td>-0.068</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.0002</td>
<td>-0.0005</td>
<td>0.3341</td>
<td>-0.5359</td>
<td>-0.1456</td>
</tr>
<tr>
<td>-0.0097</td>
<td>-0.5628</td>
<td>0.5519</td>
<td>0.1743</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0017</td>
<td>0.5402</td>
<td>-0.2088</td>
<td>-0.0437</td>
<td>-0.1033</td>
</tr>
<tr>
<td>-0.2173</td>
<td>-0.2832</td>
<td>-0.0499</td>
<td>0.009</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0002</td>
<td>-0.0083</td>
<td>0.5401</td>
<td>-0.2829</td>
<td>0.2522</td>
</tr>
<tr>
<td>0.849</td>
<td>-0.1346</td>
<td>-0.0192</td>
<td>-0.0155</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-0.0025</td>
<td>0.2094</td>
<td>0.379</td>
<td>-0.0798</td>
<td>0.1176</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

109 | Page
Table 6.7: Ordered Eigenvalues

|        | 5302.7 | 2021.2 | 376.3 | 0.9 | 0.1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 6.8: Ordered Eigenvectors

<table>
<thead>
<tr>
<th>PC1</th>
<th>PC2</th>
<th>PC3</th>
<th>PC4</th>
<th>PC5</th>
<th>PC6</th>
<th>PC7</th>
<th>PC8</th>
<th>PC9</th>
<th>PC10</th>
<th>PC11</th>
<th>PC12</th>
<th>PC13</th>
<th>PC14</th>
<th>PC15</th>
<th>PC16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3807</td>
<td>-0.1697</td>
<td>-0.0988</td>
<td>-0.8257</td>
<td>-0.2966</td>
<td>0.0339</td>
<td>-0.0402</td>
<td>0.0174</td>
<td>-0.1694</td>
<td>-0.0362</td>
<td>0.1167</td>
<td>0.0006</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.2256</td>
<td>-0.5047</td>
<td>0.7843</td>
<td>0.0608</td>
<td>-0.1586</td>
<td>-0.1008</td>
<td>-0.1483</td>
<td>-0.0308</td>
<td>0.0452</td>
<td>-0.1167</td>
<td>0.0391</td>
<td>0.0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.1433</td>
<td>-0.6572</td>
<td>-0.3383</td>
<td>0.289</td>
<td>-0.322</td>
<td>0.2816</td>
<td>0.2111</td>
<td>0.1795</td>
<td>0.273</td>
<td>0.0716</td>
<td>-0.1005</td>
<td>-0.0005</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.0152</td>
<td>-0.2315</td>
<td>-0.04</td>
<td>0.0305</td>
<td>0.2198</td>
<td>-0.1041</td>
<td>0.1838</td>
<td>-0.1006</td>
<td>-0.6069</td>
<td>0.6527</td>
<td>-0.2149</td>
<td>-0.0003</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.9855</td>
<td>0.1696</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0.3803</td>
<td>0.3125</td>
<td>0.1362</td>
<td>0.2091</td>
<td>-0.2606</td>
<td>0.1848</td>
<td>-0.4979</td>
<td>0.4519</td>
<td>-0.0378</td>
<td>0.3634</td>
<td>-0.0884</td>
<td>-0.0003</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.0839</td>
<td>0.0762</td>
<td>0.041</td>
<td>-0.2186</td>
<td>0.3895</td>
<td>-0.2299</td>
<td>0.3973</td>
<td>0.5635</td>
<td>0.4139</td>
<td>0.2689</td>
<td>0.1189</td>
<td>0.0007</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.0513</td>
<td>0.056</td>
<td>0.0249</td>
<td>-0.1915</td>
<td>0.0497</td>
<td>-0.085</td>
<td>-0.0401</td>
<td>-0.0802</td>
<td>0.2485</td>
<td>-0.0784</td>
<td>-0.9334</td>
<td>-0.0005</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0001</td>
<td>-0.0004</td>
<td>0</td>
<td>0.0011</td>
<td>0.0052</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-0.5209</td>
<td>0.0989</td>
<td>0.4571</td>
<td>-0.1456</td>
<td>-0.5359</td>
<td>0.1904</td>
<td>0.3341</td>
<td>0.1146</td>
<td>-0.005</td>
<td>0.1894</td>
<td>-0.068</td>
<td>-0.0002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-0.0113</td>
<td>0.005</td>
<td>0.0028</td>
<td>-0.1033</td>
<td>-0.0437</td>
<td>-0.0097</td>
<td>-0.2088</td>
<td>-0.5628</td>
<td>0.5402</td>
<td>0.5519</td>
<td>0.1743</td>
<td>0.0017</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.5722</td>
<td>0.3032</td>
<td>0.1238</td>
<td>0.2522</td>
<td>-0.2829</td>
<td>-0.2173</td>
<td>0.5401</td>
<td>-0.2832</td>
<td>-0.0083</td>
<td>-0.0499</td>
<td>0.009</td>
<td>0.0002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.1742</td>
<td>0.1502</td>
<td>0.1176</td>
<td>-0.0798</td>
<td>0.379</td>
<td>0.849</td>
<td>0.2094</td>
<td>-0.1346</td>
<td>-0.0025</td>
<td>-0.0192</td>
<td>-0.0155</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.9: Calculated Principal Components

<table>
<thead>
<tr>
<th>PC1</th>
<th>PC2</th>
<th>PC3</th>
<th>PC4</th>
<th>PC5</th>
<th>PC6</th>
<th>PC7</th>
<th>PC8</th>
<th>PC9</th>
<th>PC10</th>
<th>PC11</th>
<th>PC12</th>
<th>PC13</th>
<th>PC14</th>
<th>PC15</th>
<th>PC16</th>
</tr>
</thead>
<tbody>
<tr>
<td>-6.770</td>
<td>89.713</td>
<td>8.129</td>
<td>-0.009</td>
<td>-0.004</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>90.429</td>
<td>-15.185</td>
<td>-1.152</td>
<td>1.443</td>
<td>0.082</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>90.454</td>
<td>-15.547</td>
<td>-2.15</td>
<td>-1.439</td>
<td>-0.081</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-59.471</td>
<td>-2.828</td>
<td>-36.263</td>
<td>0.029</td>
<td>-0.001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-57.446</td>
<td>-28.594</td>
<td>15.897</td>
<td>0.208</td>
<td>-0.533</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-57.195</td>
<td>-27.556</td>
<td>15.545</td>
<td>-0.232</td>
<td>0.539</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Step 4: Calculating the Eigenvectors and Eigenvalues of the Covariance Matrix

Eigenvectors and eigenvalues are important, as they provide useful information (e.g., patterns in the data) about the dataset. In this case study, they are worked out using covariance matrix of the mean adjusted penetration test results (from Table 6.5) and following MATLAB’s command:

\[
\text{[eigenvector,eigenvalue]} = \text{eig(covariance_matrix)};
\]

The calculated eigenvectors are presented in Table 6.6. In general, to order the PCs according to their significance, the eigenvalues should be sorted in descending order. The ordered eigenvalues should be used to sort the eigenvectors. The ordered eigenvalues and eigenvectors are presented in Tables 6.7 and 6.8, respectively and are calculated using following MATLAB’s commands:

\[
\begin{align*}
\text{% sorting eigenvalues in descending order} \\
\text{[sorted_eigenvalue,index]} &= \text{sort(-abs(diag(eigenvalue)))}; \\
\text{% sorting eigenvectors using eigenvalues} \\
\text{sorted_eigenvector} &= \text{eigenvector(:,index)};
\end{align*}
\]

Step 5: Calculating the PCs

*PCs* are the outputs of the *PCA* and are a set of linearly uncorrelated variables created from the original observation values. In this case study, they are calculated from the multiplication of the ordered eigenvectors (from Table 6.8) and mean adjusted penetration test results (from Table 6.4) and are presented in Table 6.9.

Step 6: Choosing the PC's

This is the final step in *PCA* where the dimensionality reduction takes place. At this stage, the *PCs* with lesser significance can be ignored. There are some common rules of thumb to choose the number of *PCs* while retaining maximum information:
1. The first common rule is to choose the smallest number of PCs such that the desired percentage of explained variation is exceeded [102]. The total variance is the sum of variances of all PCs and the variance explained by a PC is the ratio between the cumulative variance of that PC and the total variance. For example, in this case study, the percentage of variance explained by PC1 and PC2 can be calculated as follows:

\[
\text{variance of PC1} = 5302.655
\]
\[
\text{variance of PC2} = 2021.133
\]
\[
\text{total variances of all PCs} = 7701.035
\]
\[
\text{percentage of variance explained by PC1} = \frac{5302.655}{7701.035} \times 100 = 68.86\%
\]
\[
\text{percentage of variance explained by PC2} = \frac{5302.655 + 2021.133}{7701.035} \times 100 = 95.10\%
\]

2. The second common approach is to use a scree graph, in which the eigenvalues are plotted against their respective component numbers. The number of PCs is then chosen so that the line in the scree graph is “steep” to its left but not to its right (elbow point) since the remaining eigenvalues (on the right-hand side of the elbow point) are relatively small and all about the same size [102].

Figure 6.14: Variance Explained by each PC
Figures 6.14 and 6.15 illustrate the plots of the percentage of variance explained by each PC (from Table 6.9) and scree graph for this case study, respectively. As presented in Figure 6.14, the first three PCs have about 100% of the variations in the penetration test results. Hence, they are selected as inputs to CA.

Recall from Section 2.5 that sometimes in literature PCA is employed to reduce the dimensions of the dataset before CA, hoping to reduce the running time for CA’s computation using a fewer number of PCs. Figure 6.16 illustrates the plot of the original penetration test results of the WSs, in this case study, against the first two calculated PCs (from Table 6.9), which have about 95% of the variations in penetration test results (see Figure 6.14). The projections of the penetration test results on the first PC (the vertical axis)
and the second PC (the horizontal axis) clearly highlight the separation between two clusters in the data, in each case (three clusters in total). It shows that PCA has successfully identified the patterns among this 16x6 dimension dataset. Hence, based on this information, the first three PCs (containing the majority of variations in penetration test results as explained previously) can be divided into three groups using CA (see next section).

6.1.7 Cluster Analysis based on Principal Component Analysis Results

As discussed previously, CA enables splitting up the data into a certain number of clusters, based on their similarities or dissimilarities [86]. Clustering approaches can be divided into two main groups of hierarchical and partitional [86]. The hierarchical approach groups data objects after a sequence of partitions, either from singleton clusters to a cluster containing all data objects or vice versa, while partitional clustering directly divides data objects into some pre-specified number of clusters without any hierarchical structure [86]. The hierarchical approaches are not suitable for large-scale datasets due to their quadratic computational complexities in both execution time and storage space [86]. However, K-means algorithm is linear in the number of data objects and for the same amount of data it will take much less amount of computational time [86]. Hence, in this framework, K-means is adopted as CA algorithm.

One of the issues in CA is to determine the number of clusters, which is called “the fundamental problem of cluster validity” by Dubes [126]. However, in this framework, the number of clusters are selected through PCA on the penetration test results of the OTSWs (as presented in the previous section) and such that 2f+1 (2f+1 replicas can tolerate f simultaneous faults) fault-tolerance condition is satisfied (enabling majority voting).

In this case study, the selected PCs are split up into three clusters (see the previous section for further information) using following MATLAB’s commands:
first_three_pcs = [-6.7709 89.7131 8.1297
                   90.4291 -15.1858 -1.1527
                   90.4547 -15.5472 -2.1560
                   -59.4717 -2.8288 -36.2632
                   -57.4461 -28.5946 15.8972
                   -57.1951 -27.5567 15.5451];

[idx,C,sumd,D] = kmeans(first_three_pcs,3);

In the above MATLAB command idx vector, C matrix, sumd vector
and D matrix contain cluster indices for each WS, centroid location of each
cluster, within-cluster sums of point-to-centroid distances and distances of
each WS to every cluster’s centroid, respectively.

The result of WSs clustering based on their security vulnerabilities di-
versity and the distances of each WS to the centre of each cluster are shown
in Tables 6.10 and 6.11, respectively.

Table 6.10: Results of CA on selected PCs

<table>
<thead>
<tr>
<th>WS</th>
<th>Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>3</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>2</td>
</tr>
<tr>
<td>S5</td>
<td>2</td>
</tr>
<tr>
<td>S6</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6.11: Distance between Each WS and the Cluster Centres (in 1.0e+04 scales)

<table>
<thead>
<tr>
<th>WS</th>
<th>Cluster 1</th>
<th>Cluster 2</th>
<th>Cluster 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>2.0588</td>
<td>1.4686</td>
<td>0.0000</td>
</tr>
<tr>
<td>S2</td>
<td>0.0000</td>
<td>2.2063</td>
<td>2.0538</td>
</tr>
<tr>
<td>S3</td>
<td>0.0000</td>
<td>2.2067</td>
<td>2.0638</td>
</tr>
<tr>
<td>S4</td>
<td>2.3829</td>
<td>0.1486</td>
<td>1.3312</td>
</tr>
<tr>
<td>S5</td>
<td>2.2354</td>
<td>0.0387</td>
<td>1.6625</td>
</tr>
<tr>
<td>S6</td>
<td>2.2241</td>
<td>0.0357</td>
<td>1.6350</td>
</tr>
</tbody>
</table>

Recall that the objective of the service selection framework in this di-
sertation is to select group of OTSWSs with security vulnerabilities diversity
(identified through penetration testing) for implementation of ITWS. And to
facilitate this service selection approach, OTSWSs should be clustered ac-
cording to their security vulnerabilities similarity, so that groups of services
with security vulnerabilities diversity can be formed based on services se-
lected from these clusters (one service from each of the resultant clusters).
Hence, using the CA results presented in Table 6.10, the WSs’ are grouped such that each group consists of one service from each of the clusters:

- **Group1**: S1, S2 and S4
- **Group2**: S1, S2 and S5
- **Group3**: S1, S2 and S6
- **Group4**: S1, S3 and S4
- **Group5**: S1, S3 and S5
- **Group6**: S1, S3 and S6

The next section explains the process of ordering these groups according to their overall security vulnerabilities.

### 6.1.8 WS-Groups Ordering using Penetration Testing

The final step in this service selection framework is about performing penetration testing on each group of services to identify their overall security vulnerabilities and using this information to order the groups so that the ITWS could be started with the most secure group and switch to the next group if the first group fails and so on.

![Component Diagram for Penetration Testing Each Group of WSs](image)
Figure 6.17 shows the component diagram for this step indicating that the ITWS (called CompositeWS in this figure) and WS-Attacker consist of OTSWs and attack plugins, respectively and that the ITWS can be tested through WS-Attacker’s interface.

Each WS-group is developed through the composition of their associated WSs, using BPEL 2.0 plugin for Eclipse Neon 2 4.6.2. And is deployed on Apache 1.3.4 runtime (on top of Tomcat 5.5.26 server) running on the same machine that is hosting the WSs (Intel Core™ i5-3320M CPU @ 2.60GHz system with 7.88GB usable RAM and 64-bit Operating System). The operation of these WS-groups was as follows:

1. Receives client’s request.
2. Passes the client’s request to each of the parallel invocation processes (each process invokes one of the three WSs associated with the group).
3. Each concurrent invocation process invokes its associated WS.
4. The business process replies to the client once a response is returned from each of the invoked WSs.

For ordering the WS-groups, Coercive Parsing, DJBX31A Hash Collision, and XML Attribute Count attacks are performed on each group. Table 6.12 presents the WS-Attacker’s settings for each of these tests. In these tests, the attack element was either plotted in the header or the body of the test SOAP message as presented in Message 5.1 and Message 5.2, respectively.

Table 6.12: WS-Attacker’s Settings

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Test Message Settings</th>
<th>WS-Attacker’s other Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>Coercive Parsing attack with 20,000 open tags plotted in the header of the SOAP message</td>
<td>2 parallel attack threads, 4 requests per thread, 9999 milliseconds between every testprobe request, 750 milliseconds between every attack request, 4 seconds server recovery time, 5 seconds stop after the last tampered request</td>
</tr>
<tr>
<td>Test 2</td>
<td>DJBX31A Hash Collision attack with 4,000 paired keys/values plotted in the body of the SOAP message</td>
<td></td>
</tr>
<tr>
<td>Test 3</td>
<td>XML Attribute Count attacks with 60,000 paired keys/values plotted in the body of the SOAP message</td>
<td></td>
</tr>
</tbody>
</table>

Figures 6.18-6.23 show the penetration test results for test attacks performed on these groups.
Figure 6.18: Penetration Test Results for WS-Group1 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively)

Figure 6.19: Penetration Test Results for WS-Group2 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively)
Figure 6.20: Penetration Test Results for WS-Group3 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively)

Figure 6.21: Penetration Test Results for WS-Group4 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively)
Figure 6.22: Penetration Test Results for WS-Group5 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively)

Figure 6.23: Penetration Test Results for WS-Group6 (for information about each test and WS-group see Table 6.12 and Section 6.1.7, respectively)
The penetration test results show the vulnerability of these WS-groups to the performed attacks in percentage so that 100% indicates that the group is very vulnerable to the attack and 1% shows that it does not have this security vulnerability.

Each test is repeated ten (to increase the confidence of the outcome of these tests) times and the average of the results is considered. Because, these test plugins decide on the security vulnerability of the WSs based on their response times that slightly differs every time the same test is performed. The vertical and horizontal lines in these graphs (Figures 6.18-6.23) show the result of the penetration test every time the same test is performed and the average of the penetration test results for each test, respectively.

Finally, the WS-groups are ordered according to their penetration test result for each test (see Figure 6.24). For example, Group2 has the lowest penetration test result in each test, so it is the most secure group among others. The results for XML Attribute Count test are omitted in these comparisons as all these WS-groups have similar level of vulnerability to this attack (see Figures 6.18-6.23).

The penetration test results show the vulnerability of these WS-groups to the performed attacks in percentage so that 100% indicates that the group is very vulnerable to the attack and 1% shows that it does not have this security vulnerability.
Considering the results presented in Figures 6.24a and 6.24b the WS-groups can be classified according to their overall security vulnerabilities (in descending order) as follows:

- **Group2**: S1, S2 and S5
- **Group3**: S1, S2 and S6
- **Group5**: S1, S3 and S5
- **Group6**: S1, S3 and S6
- **Group1**: S1, S2 and S4
- **Group4**: S1, S3 and S4

### 6.2 Summary

This chapter illustrated the integration of the penetration test results of the WSs, PCA, and CA in the selection of security-aware WSs for implementation of ITWSs (the proposed service selection framework). It also explained why these approaches are adopted. The result of this case study is utilized in Chapter 8 to evaluate this framework.
CHAPTER 7

Dynamic Reconfiguration of ITWS Using BPEL and JAVA as BPEL Extension

Chapter 6 demonstrated the proposed service selection framework through a case study. This chapter uses different sets of case studies to illustrate the feasibility of implementing self-reconfigurable Intrusion-Tolerant Web Service (ITWSs) utilizing a combination of Business Process Engineering Language (BPEL) constructs and Java as BPEL extension, as well as using Java as BPEL extension only.

Section 7.1 presents the common setups for these case studies. Sections 7.2 and 7.3 demonstrate the implementation of self-reconfigurable ITWSs using a combination of BPEL constructs and Java as BPEL extension approach and utilizing only Java as BPEL extension approach, respectively. Finally, Section 7.4 summarizes the work presented in this chapter.

7.1 Setups for Case Studies

This section presents the common setups for the case studies demonstrated in this chapter.

7.1.1 WS Preparation

Recall that in service selection case study OTSWSs were not employed because we did not have permission to perform penetration tests on such services. However, the case studies presented in this chapter did not require to perform any penetration test on the involved WSs, and moreover, it was a good practice to adopt as many of OTSWSs as possible. For these
case studies the following simple calculator Web Services (WSs) are developed using the source code presented in Code 5.1 (see Chapter 5):

- **WS** developed using *Apache Axis2* 1.5.1 framework and was deployed on *Apache Tomcat* 6.0.18 running on *Intel® Core™ i5-3320M CPU* @ 2.60GHz system with 7.88GB usable *RAM* and 64-bit Operating System.

- **WS** developed using *Apache Axis2* 1.5.1 framework and was deployed on *Apache Tomcat* 6.0.18 running on *Intel® Xeon® CPU E3-1240 V2* @ 3.40GHz system with 16.0GB *RAM* and 64-bit Operating System.

- **WS** developed using *NET 4.0* framework and was deployed on *Intel® Core™ 2 Duo CPU* P8400@ 2.26GHz system with 3.00GB *RAM* and 32-bit Operating System.

Also, two third-party *ASP.NET WSs* [127], [128], which provide similar calculator services, are employed.

### 7.1.2 DB Preparation

For these case studies, *Oracle Database* 12.1.0.1 is utilized and Table 7.1 is created to store the necessary information for invocation of the ITWS’s constituent WSs. The *SERVICEGROUP* and *PRIORITY* are equivalent to *invocationId* and *orderNumber* properties in the proposed architecture (see Chapter 3 for further information), respectively.

Table 7.1: Service Table Storing Necessary Information for Invoking Candidate WSs

<table>
<thead>
<tr>
<th>COLUMNS</th>
<th>DATA TYPE</th>
<th>NULLABLE</th>
<th>DATA_DEFAULT</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ID</td>
<td>VARCHAR2 (5 BYTE)</td>
<td>No</td>
<td>(null)</td>
<td>1 (null)</td>
</tr>
<tr>
<td>2 NAME</td>
<td>VARCHAR2 (50 BYTE)</td>
<td>Yes</td>
<td>(null)</td>
<td>2 (null)</td>
</tr>
<tr>
<td>3 ENDPOINT</td>
<td>VARCHAR2 (200 BYTE)</td>
<td>Yes</td>
<td>(null)</td>
<td>3 (null)</td>
</tr>
<tr>
<td>4 TARGETNAMESPACE</td>
<td>VARCHAR2 (200 BYTE)</td>
<td>Yes</td>
<td>(null)</td>
<td>4 (null)</td>
</tr>
<tr>
<td>5 SERVICEGROUP</td>
<td>VARCHAR2 (50 BYTE)</td>
<td>Yes</td>
<td>(null)</td>
<td>5 (null)</td>
</tr>
<tr>
<td>6 PRIORITY</td>
<td>NUMBER (30.0)</td>
<td>Yes</td>
<td>(null)</td>
<td>6 (null)</td>
</tr>
<tr>
<td>7 OPERATION</td>
<td>VARCHAR2 (50 BYTE)</td>
<td>Yes</td>
<td>(null)</td>
<td>7 (null)</td>
</tr>
<tr>
<td>8 INPUT</td>
<td>VARCHAR2 (50 BYTE)</td>
<td>Yes</td>
<td>(null)</td>
<td>8 (null)</td>
</tr>
<tr>
<td>9 OUTPUT</td>
<td>VARCHAR2 (50 BYTE)</td>
<td>Yes</td>
<td>(null)</td>
<td>9 (null)</td>
</tr>
</tbody>
</table>
7.1.3 Communication with DB

In these studies, the dynamic communication between the BPEL process and the Database (DB), for collection of the information required for invocation of the WSs associated with each group, is done through Java as BPEL extension and using Java SQL library (see Appendix G for complete Java class). The overview of this communication is as follows:

1. Receives from the BPEL process the information (invocationId and orderNumber) about the group of WSs that should be invoked.
2. Stablishes connection with the DB.
3. Runs:

   "SELECT ENDPOINT, TARGETNAMESPACE, OPERATION, INPUT FROM HR.WEBSERVICES "+ "WHERE SERVICEGROUP =" + invocationId + "AND PRIORITY = " + orderNumber);

   "SELECT PRIORITY FROM HR.WEBSERVICES";

4. From the results of the above queries, it prepares a string (the format that could be used by the business process) containing the information required for invoking the WSs associated with that group, the number of WSs within that group, and the maximum number of groups that are available to be used at that particular part of the business process. From the last information the business process would understand if there are any other groups available that could be used if the current group fails.
5. It returns the prepared information to the business process.

7.2: Case Study: Dynamic Reconfiguration Using a Combination of Java as BPEL Extension and BPEL Constructs

This section uses a case study to demonstrate the feasibility of implementing self-reconfigurable ITWSs utilizing a combination of Java as BPEL extension and BPEL constructs. For this study, a self-reconfigurable ITWS is developed using the WSs from the Section 7.1.1 and BPEL plugin for Oracle SOA Suite 12c running on Intel® Core™ i5-3320M CPU @
2.60GHz system with 7.88GB usable RAM and 64-bit Operating System. Figure 7.1 presents the BPEL diagram for this ITWS.

Figure 7.1: BPEL Diagram of Dynamically Reconfigurable ITWS, Implemented Using a Combination of Java as BPEL Extension and BPEL Constructs
The overview of the operation of the self-reconfigurable ITWS presented in Figure 7.1 is as follows:

1. Receives the client’s request (start of the business process).
2. Stores the start time of the business process, which will be used to work out the response time of this self-reconfigurable ITWS.
3. Executes the Java class (from Section 7.1.3) responsible for retrieving, from DB, the information required for invoking the WSs associated with the WS-group that should be used (see Code J.1, in Appendix J).
4. Creates parallel process paths, using ForEachN activity and the number of WSs associated with that group returned by the DB (see Section 7.1.3), for invoking the WSs from the previous step.
5. Each WS invocation path (created in the last step) uses the Code J.2 (see Appendix J) to execute the Java class (see Appendix H), which utilizes Java’s API for XML-based RPC (JAX-RPC) [129] for dynamic invocation of its associated WS. The advantage of using a Dynamic Invocation Interface is that WSs can be invoked without the need for any pre-runtime information or any static operations (e.g., WSDL to Java mapping).
6. Checks whether sufficient (for performing majority voting) responses are returned by the WSs.
   - If adequate responses are returned, it performs majority voting on the BPEL’s local variables that hold these responses, using the Code J.3 (see Appendix J) then executes step 7.
   - Else, it throws an exception
     - BPEL’s CatchAll construct catches the thrown exception.
     - It uses Code J.4 (see Appendix J) to check whether an alternative group of services is available by comparing the orderNumber (see Chapter 3 for more information)

---

5 Oracle SOA Suite 12c offers ForEachN activity, which creates parallel process paths at runtime [111]. This activity enables to dynamically implement N-version programming when the number of constituent WSs is only known at runtime.
for the current group and the maximum number of available groups provided by the \textit{DB} (see Section 7.1.3).

- If an alternative group is available, it updates the \textit{BPEL}'s local variable storing the \textit{orderNumber} for the group that should be used next and re-executes the \textit{ITWS} from step 3
- Else, it throws an exception and terminates the \textit{BPEL} process.

7. Assigns the result of successful execution of the \textit{ITWS} to the output variable.

8. Uses Code J.6 (see Appendix J) to execute the Java class (Code J.5 in Appendix J) that writes the start and end times of the process into a file (to work out the response time of the process for evaluation purpose).

9. Returns the result of the execution of the \textit{ITWS} to the client.

This \textit{ITWS} is run once with three and once with five \textit{WS}s and the response times of these executions are recorded (for evaluation purpose). These executions are repeated forty nine times (to increase the confidence on the evaluation results) and the average of the response times is considered as the response time of the process in each case. Because, the response time slightly varies every time the \textit{ITWS} is executed.

Figures 7.2 and 7.3 present the response times for the execution of this \textit{ITWS} running with three and five \textit{WS}s, respectively. The vertical and horizontal lines in these graphs show the response time (ms) every time the \textit{ITWS} is executed and the average of the response times (ms) respectively. These results will be used in Chapter 8 to evaluate the work presented in this dissertation.
Figure 7.2: Process Execution Time (ms) for Dynamic ITWS Implemented using Combination of Java as BPEL Extension and BPEL constructs (ran with three diverse WSs)

Figure 7.3: Process Execution Time (ms) for Dynamic ITWS Implemented using Combination of Java as BPEL Extension and BPEL constructs (ran with five diverse WSs)
7.3: Case Study: Dynamic Reconfiguration Using Only Java as BPEL Extension

Figure 7.4: BPEL Diagram of Dynamic Reconfigurable ITWS, Implemented Using a Combination of Java as BPEL Extension and BPEL Constructs
This section uses a case study to demonstrate the feasibility of implementing self-reconfigurable ITWSs utilizing Java as BPEL extension only. For this study, a self-reconfigurable ITWS is developed using the WSs from the Section 7.1.1 and BPEL plugin for Oracle SOA Suite 12c running on Intel® Core™ i5-3320M CPU @ 2.60GHz system with 7.88GB usable RAM and 64-bit Operating System. Figure 7.4 presents the BPEL diagram for this ITWS.

The overview of the operation of the self-reconfigurable ITWS presented in Figure 7.4 is as follows:

1. Receives the client’s request (start of the business process).
2. Stores the start time of the business process, which will be used to work out the response time of this self-reconfigurable ITWS.
3. Executes the Java class (from Section 7.1.3) responsible for retrieving, from DB, the information required for invoking the WSs associated with the WS-group that should be used (see Code J.7, in Appendix J).
4. Uses Code J.8 (see Appendix J) to execute the Java class (see Appendix I) responsible for dynamic parallel invocation of the WSs from last step. This dynamic invocation approach utilizes Java multithreading library.
5. Checks whether sufficient (for performing majority voting) responses are returned by the WSs.
   - If adequate responses are returned, it performs majority voting on the BPEL’s local variables that hold these responses, using the Code J.3 (see Appendix J) then executes step 6.
   - Else, it throws an exception
     - BPEL’s CatchAll construct catches the thrown exception.
     - It uses Code J.4 (see Appendix J) to check whether an alternative group of services is available by comparing the orderNumber (see Chapter 3 for more information)
for the current group and the maximum number of available groups provided by the DB (see Section 7.1.3).

- If an alternative group is available, it updates the BPEL’s local variable storing the orderNumber for the group that should be used next and re-executes the ITWS from step 3
- Else, it throws an exception and terminates the BPEL process.

6. Assigns the result of successful execution of the ITWS to the output variable.
7. Uses Code J.6 (see Appendix J) to execute the Java class (Code J.5 in Appendix J) that writes the start and end times of the process into a file (to work out the response time of the process for evaluation purpose).
8. Returns the result of the execution of the ITWS to the client.

This ITWS is run once with three and once with five WSs and the response times are recorded (for evaluation purpose). These executions are repeated forty nine times (to increase the confidence of the outcome of these tests) and the average of the response times is considered as the response time of the process in each case. Because, the response time slightly varies every time the ITWS is executed.

Figures 7.5 and 7.6 present the response times for the execution of this ITWS running with three and five WSs, respectively. The vertical and horizontal lines in these graphs show the response time (ms) every time the ITWS is executed and the average of the response times (ms) respectively. These results will be used in Chapter 8 to evaluate the work presented in this dissertation.
Figure 7.5: Process Execution Time (ms) for Dynamic ITWS Implemented using Java as BPEL Extension only (ran with three diverse WSs)

<table>
<thead>
<tr>
<th>1st run</th>
<th>2nd run</th>
<th>3rd run</th>
<th>4th run</th>
<th>5th run</th>
<th>6th run</th>
<th>7th run</th>
<th>8th run</th>
<th>9th run</th>
<th>Avg Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.01</td>
<td>1.00</td>
<td>1.04</td>
<td>1.02</td>
<td>1.03</td>
<td>1.01</td>
<td>1.02</td>
<td>1.03</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>1.05</td>
<td>1.03</td>
<td>1.04</td>
<td>1.02</td>
<td>1.03</td>
<td>1.01</td>
<td>1.02</td>
<td>1.03</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>1.07</td>
<td>1.04</td>
<td>1.05</td>
<td>1.03</td>
<td>1.04</td>
<td>1.02</td>
<td>1.03</td>
<td>1.04</td>
<td>1.03</td>
<td>1.03</td>
</tr>
</tbody>
</table>

Figure 7.6: Process Execution Time (ms) for Dynamic ITWS Implemented using Java as BPEL Extension only (ran with five diverse WSs)

<table>
<thead>
<tr>
<th>1st run</th>
<th>2nd run</th>
<th>3rd run</th>
<th>4th run</th>
<th>5th run</th>
<th>6th run</th>
<th>7th run</th>
<th>8th run</th>
<th>9th run</th>
<th>10th run</th>
<th>Avg Execution Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.02</td>
<td>1.03</td>
<td>1.04</td>
<td>1.05</td>
<td>1.06</td>
<td>1.07</td>
<td>1.08</td>
<td>1.09</td>
<td>1.10</td>
<td>1.11</td>
<td>1.09</td>
</tr>
<tr>
<td>1.01</td>
<td>1.02</td>
<td>1.03</td>
<td>1.04</td>
<td>1.05</td>
<td>1.06</td>
<td>1.07</td>
<td>1.08</td>
<td>1.09</td>
<td>1.10</td>
<td>1.07</td>
</tr>
<tr>
<td>1.05</td>
<td>1.06</td>
<td>1.07</td>
<td>1.08</td>
<td>1.09</td>
<td>1.10</td>
<td>1.11</td>
<td>1.12</td>
<td>1.13</td>
<td>1.14</td>
<td>1.10</td>
</tr>
</tbody>
</table>

Average of Process Execution Times (ms): 1.06
7.4: Summary

This chapter illustrated the feasibility of implementing self-reconfigurable ITWSs using a combination of BPEL constructs and Java as BPEL extension, as well as utilizing Java as BPEL extension only, through two case studies. The results of these case studies are utilized in Chapter 8 to evaluate this framework.
CHAPTER 8

Evaluation

Previous chapters have presented the background and motivations, the proposed framework architecture, the service selection framework and possible dynamic Business Process Engineering Language (BPEL) process reconfiguration for the work presented in this dissertation. This chapter uses the outcomes of the case studies presented in Chapters 4-7 to evaluate the proposed solutions in terms of the central question underlying this work:

Can a self-reconfigurable Intrusion-Tolerant Web Service (ITWS), implemented using N-version programming and diversity formed by composing Off-The-Shelf Web Services (OTWSs) that are selected through penetration testing, Principal Component Analysis (PCA), and Cluster Analysis (CA) processes mitigate XML-related security vulnerabilities?

Section 8.1 evaluates the advantages, and disadvantages of the ITWS presented in this dissertation (as a countermeasure against XML-related cyber-attacks) over other existing countermeasure approaches. Section 8.2 evaluates the ITWS; formed based on penetration test results of candidate OTWSs, in terms of mitigating the XML-related security vulnerabilities. Section 8.3 evaluates the effectiveness of PCA and CA utilization in security-aware service selection based on penetration tests results of the candidate OTWSs. Section 8.4 evaluates the utilization of Java as BPEL extension in the implementation of self-reconfigurable ITWS.
8.1 Advantages and Limitations of the Presented ITWS

This section evaluates the ITWS framework presented in this dissertation in terms of its advantages, limitations, and extensibility as a countermeasure against XML-related cyber-attacks.

8.1.1 Advantages of the Presented ITWS

Compared to the disadvantages of existing countermeasures against XML-related cyber-attacks (presented in Chapter 2), this ITWS has the following advantages:

➢ It is independent of any WS-* security standard, hence:
  o It does not require any changes to the structure of the messages.
  o Revocation, limitations or improper utilization of such standards does not affect this framework.
➢ It does not require schema validation, hence:
  o It does not cause high CPU load and large memory consumption.
  o It does not require any arbitrary restriction on the number and length of the namespaces, which eliminates the unpredictable rejection of messages.
  o It does not require manual schema creation and/or update that may make Web Services (WSs) prone to cyber-attacks.
  o It does not introduce security vulnerabilities as a result of a loosely defined schema.
  o It is not prone to the cyber-attacks targeting a schema validator.

8.1.2 Extensibility of the Presented ITWS

The presented ITWS is extensible so that diversity can also be applied to penetration testing tools and tests. Furthermore, automation can be
applied to its service discovery stage, which is currently the responsibility of
the system’s administrator.

8.1.3 Limitations of the Presented ITWS

However, this ITWS has the following limitations:

- There may not be enough number of OTSWs to implement the ITWS.
- Penetration test results may differ if they were performed in the actual operational environment.
- The ITWS is wrapped in a BPEL process, which itself may be a single point of failure.
- Recall that SM tests all OTSWs every $t+i$ time unit (see Chapter 3), the security vulnerabilities of these services may change during these time units, which invalidates the existing service groups. However, this limitation is inevitable due to the dynamic nature of WS’s operating environment. But this effect may be diminished by selecting $i$ as smallest time unit that the ITWS system can support.

8.2 Feasibility of Implementing ITWS Based on Penetration Testing Results of Candidate WSs

- **Research Question One:** Does an ITWS, formed based on penetration test results of candidate WSs, mitigate XML-related security vulnerabilities?

Chapter 4 used a case study to demonstrate the feasibility of an ITWS implemented based on the penetration test results of its constituent WSs. In this case study, the ITWS was implemented using four WSs one of them having security vulnerability to *Coercive Parsing DoS* attack while other three WSs did not have this security vulnerability. The outcome of this study showed that the resultant ITWS also did not have this security vulnerability (see Chapter 4 for further details). Hence:
The answer to the research question one is yes, an ITWS, formed based on penetration testing results of candidate WSs, mitigates XML-related security vulnerabilities.

8.3 Feasibility of PCA and CA Utilization in Security-Aware Service Selection

- **Research Question Two**: Is PCA an effective pre-processing step to CA in this service selection framework?
- **Research Question Three**: Do PCA and CA improve the process of security-aware service selection based on penetration testing results of candidate WSs?
- **Research Question Four**: Does an ITWS in which N-version programming and diversity (formed by composing OTSWSs selected through PCA and CA analysis on their penetration testing results) are used, mitigate XML-related security vulnerabilities?

As demonstrated in Section 6.1.6 (Chapter 6), PCA has successfully reduced the dimensionality of the penetration test results of the WSs (from Section 6.1.2) and the computation complexity of CA. It also has identified the patterns among the penetration test results of these WSs and has enabled to determine the number of clusters, which is one of the issues with CA. Hence, it has proved to be a practical pre-CA approach in this service selection framework. Therefore,

The answer to the research question two is yes, PCA is an effective pre-processing step to CA in this service selection framework.

Recall that CA assigned S4, S5, and S6 to the same cluster (see Section 6.1.7), which means that these WSs have the minimum security vulnerabilities diversity among the services from Section 6.1.2. Hence, to evaluate this work in terms of research questions three and four (see above), the WS-groups formed in Section 6.1.7 are compared with a group containing S4, S5 and S6 (hereafter referred to as WS-group7).
Figure 8.1: Penetration Test Results for WS-Group7 (for information about each test see Table 6.12)

Figure 8.2: Average of Penetration Test Results for WS-Groups1-6 against Average of Penetration Test Results for WS-Group7 (for information about each test see Table 6.12)
WS-group7 is also developed as it is explained in Section 6.1.8 and is tested using Coercive Parsing, DJBX31A Hash Collision, and XML Attribute Count attacks with WS-Attacker’s settings presented in Table 6.12. In these tests, the attack element was either plotted in the header or the body of the test SOAP message as presented in Message 5.1 and Message 5.2, respectively.

Figure 8.1 shows the results of the test attacks performed on WS-group7. These results show the vulnerability of this WS-group to the performed attacks in percentage so that 100% indicates that the group is very vulnerable to the attack and 1% shows that it does not have this security vulnerability.

Each test is repeated ten times (to increase the confidence of the outcome of these tests), and the average of the test results is considered as the final test result since these test plugins decide on the security vulnerability of the WSs based on their response times that slightly differs every time the same test is performed. The vertical and horizontal lines in these graphs (Figures 8.1a-8.1c) show the result of the penetration test every time the same test is performed and the average of the penetration test results for each test, respectively.

Figure 8.2 presents the average penetration test results of WS-groups1-6 against the average penetration test results of WS-group7 for Coercive Parsing and Hash Collision attacks. The results of XML Attribute Count attack test are excluded in this comparison as its result is similar for all these WS-groups. Comparing the penetration test results of WS-group7 (non-optimal group) with the optimal WS-groups (identified through PCA and CA) show 17.96%-29.36% and 0.61%-3.06% better overall security vulnerability to Coercive Parsing and Hash Collision attacks, respectively.
Table 8.1: 2-Samples Tests Results for Coercive Parsing Attack

<table>
<thead>
<tr>
<th>Null hypothesis</th>
<th>Alternative hypothesis</th>
<th>Confidence interval</th>
<th>p-value</th>
<th>Null hypothesis rejected?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group7 and Group1 are equally vulnerable to Coercive Parsing attack</td>
<td>Group7 is more vulnerable to Coercive Parsing attack than Group1</td>
<td>95%</td>
<td>0.000</td>
<td>Yes</td>
</tr>
<tr>
<td>Group7 and Group2 are equally vulnerable to Coercive Parsing attack</td>
<td>Group7 is more vulnerable to Coercive Parsing attack than Group2</td>
<td>95%</td>
<td>0.000</td>
<td>Yes</td>
</tr>
<tr>
<td>Group7 and Group3 are equally vulnerable to Coercive Parsing attack</td>
<td>Group7 is more vulnerable to Coercive Parsing attack than Group3</td>
<td>95%</td>
<td>0.000</td>
<td>Yes</td>
</tr>
<tr>
<td>Group7 and Group4 are equally vulnerable to Coercive Parsing attack</td>
<td>Group7 is more vulnerable to Coercive Parsing attack than Group4</td>
<td>95%</td>
<td>0.000</td>
<td>Yes</td>
</tr>
<tr>
<td>Group7 and Group5 are equally vulnerable to Coercive Parsing attack</td>
<td>Group7 is more vulnerable to Coercive Parsing attack than Group5</td>
<td>95%</td>
<td>0.000</td>
<td>Yes</td>
</tr>
<tr>
<td>Group7 and Group6 are equally vulnerable to Coercive Parsing attack</td>
<td>Group7 is more vulnerable to Coercive Parsing attack than Group6</td>
<td>95%</td>
<td>0.000</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 8.2: 2-Samples Tests Results for Hash Collision Attack

<table>
<thead>
<tr>
<th>Null hypothesis</th>
<th>Alternative hypothesis</th>
<th>Confidence interval</th>
<th>p-value</th>
<th>Null hypothesis rejected?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group7 and Group1 are equally vulnerable to Hash Collision attack</td>
<td>Group7 is more vulnerable to Hash Collision attack than Group1</td>
<td>95%</td>
<td>0.159</td>
<td>No</td>
</tr>
<tr>
<td>Group7 and Group2 are equally vulnerable to Hash Collision attack</td>
<td>Group7 is more vulnerable to Hash Collision attack than Group2</td>
<td>95%</td>
<td>0.092</td>
<td>No</td>
</tr>
<tr>
<td>Group7 and Group3 are equally vulnerable to Hash Collision attack</td>
<td>Group7 is more vulnerable to Hash Collision attack than Group3</td>
<td>95%</td>
<td>0.085</td>
<td>No</td>
</tr>
<tr>
<td>Group7 and Group4 are equally vulnerable to Hash Collision attack</td>
<td>Group7 is more vulnerable to Hash Collision attack than Group4</td>
<td>95%</td>
<td>0.315</td>
<td>No</td>
</tr>
<tr>
<td>Group7 and Group5 are equally vulnerable to Hash Collision attack</td>
<td>Group7 is more vulnerable to Hash Collision attack than Group5</td>
<td>95%</td>
<td>0.083</td>
<td>No</td>
</tr>
<tr>
<td>Group7 and Group6 are equally vulnerable to Hash Collision attack</td>
<td>Group7 is more vulnerable to Hash Collision attack than Group6</td>
<td>95%</td>
<td>0.107</td>
<td>No</td>
</tr>
</tbody>
</table>
In Statistics, 2-samples t-test is a type of hypothesis test that allows comparing the means of two independent groups of observations. To further assess the proposed service selection framework; 2-samples t-tests are carried out on the penetration test results of WS-group7 and each of the other WS-groups (see Tables 8.1 and 8.2). These tables show the null and alternative hypothesis as well as the confidence level for these 2-samples tests. They also illustrate the p-values and indicate whether the null hypothesis is rejected (decided based on the p-value) in each case.

For 95% confidence level, the p-value should be 0.05 or less for null hypothesis to be accepted. Therefore, as shown in Tables 8.1 and 8.2, WS-group7 is more vulnerable to Coercive Parsing attack compared with WS-groups1-6. However, all these WS-groups are similarly vulnerable to Hash Collision attack. Hence,

*The answer to the research question four is yes, an ITWS in which N-version programming and diversity (formed by composing OTSWSs selected through PCA and CA analysis on their penetration test results) are used, mitigates XML-related security vulnerabilities.*

However, the above conclusion also requires comparison of the penetration test results of WS-groups1-6 with the WS-groups formed using remaining combinations of these WSs (identified by PCA and CA as non-optimal WS-groups according to their overall XML-related security vulnerabilities).

Finally, without utilization of PCA and CA, every combination of these WSs \(\binom{m!}{n!(m-n)!}\) had to be tested in order to select the most secure group. However, these approaches have reduced the number of required penetration tests significantly. For example, in this case study six WSs are available, and three should be selected so in the normal case, twenty different WS-groups had to be tested, whereas through utilization of PCA and CA this number is reduced to six WS-groups (70% less penetration testing). Therefore,
The answer to the research question three is yes, PCA and CA improve the process of security-aware service selection based on penetration test results of candidate WSs.

8.4 Feasibility of Implementing Self-Reconfigurable ITWS Using BPEL and JAVA as BPEL Extension

Figure 8.3: BPEL Diagram of Static ITWS Implemented for Evaluation Purpose
Research Question Five: Does the use of Java as BPEL extension enables dynamic reconfiguration of ITWS?

Chapter 7 demonstrated the implementation of dynamically reconfigurable ITWS using a combination of Java as BPEL extension and BPEL constructs, as well as using only Java as BPEL extension. Hence, to evaluate this work in terms of research question five (see above), a static ITWS is implemented and is compared with the dynamic ITWSs from Section 7.

This static ITWS is developed using the WSs from the Section 7.1.1 and BPEL plugin for Oracle SOA Suite 12c running on Intel® Core™ i5-3320M CPU @ 2.60GHz system with 7.88GB usable RAM and 64-bit Operating System. Figure 8.3 presents the BPEL diagram for this ITWS. The overview of the operation of this ITWS is as follows:

1. Receives the client’s request (start of the business process).
2. Stores the start time of the business process, which will be used to work out the response time of this self-reconfigurable ITWS.
3. Invokes the WSs (from Section 7.1.1) concurrently. Each of the process paths of the flow activity invokes one of these WSs that is assigned to it at design time.
4. Checks whether sufficient (for performing majority voting) responses are returned by the WSs.
   - If adequate responses are returned, it performs majority voting on the BPEL’s local variables that hold these responses, using the Code J.3 (see Appendix J) then executes step 5.
   - Else, it throws an exception
     - BPEL’s CatchAll construct catches the thrown exception.
     - Checks if the pre-defined (by the developer of this ITWS) number of re-execution of the invocation process it met.
     - Re-executes the ITWS from step 3, if the pre-defined number of re-execution of the invocation process is not met.
- Else, it throws an exception and terminates the BPEL process.

5. Assigns the result of successful execution of the ITWS to the output variable.

6. Uses Code J.6 (see Appendix J) to execute the Java class (Code J.5 in Appendix J) that writes the start and end times of the process into a file (to work out the response time of the process for evaluation purpose).

7. Returns the result of the execution of the ITWS to the client.

This ITWS is executed once with three and once with five WSs and its response times are recorded. These executions are repeated forty nine times (to increase the confidence of the outcome of these tests) and the average of their response times is considered as the response time of the ITWS in each case. Because, the response time slightly varies every time the ITWS is executed.

Figures 8.4 and 8.5 present the response times for the executions of this ITWS running with three and five WSs, respectively. The vertical and horizontal lines in these graphs show the response time (ms) every time the ITWS is executed and the average of these executions’ response times (ms), respectively.
Figure 8.4: Process Execution Time (ms) for Static ITWS Implemented using BPEL Constructs only (ran with three diverse WSs from Section 7.1.1)

Figure 8.5: Process Execution Time (ms) for Static ITWS Implemented using BPEL Constructs only (ran with five diverse WSs from Section 7.1.1)
Figures 8.6a and 8.6b illustrate the average response times for the executions of the static ITWS against the average response times for the executions of the dynamic ITWSs ran with three and five WSs. Comparing these results indicates that the introduction of dynamic reconfiguration has introduced execution time overheads (see Table 8.3).

### Table 8.3: Execution Time Overheads

<table>
<thead>
<tr>
<th>Business Processes</th>
<th>Execution time overhead for ITWS ran with three WSs</th>
<th>Execution time overhead for ITWS ran with five WSs</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPEL + Java vs BPEL constructs only</td>
<td>44%</td>
<td>11%</td>
</tr>
<tr>
<td>Java only vs BPEL constructs only</td>
<td>62.29%</td>
<td>53.45%</td>
</tr>
<tr>
<td>BPEL + Java vs Java only</td>
<td>12.70%</td>
<td>38.25%</td>
</tr>
</tbody>
</table>

To further examine these ITWSs, 2-samples t-tests are carried out on their process execution times (see Tables 8.3 and 8.4). These tables show the null and alternative hypothesis as well as the confidence level for these 2-samples tests. They also illustrate the p-values and indicate whether the null hypothesis is rejected (decided based on the p-value) in each case.

For 95% confidence level, the p-value should be 0.05 or less to accept the null hypothesis. Therefore, as shown in Tables 8.3 and 8.4, the utilization of Java as BPEL extension to implement self-reconfigurable ITWS introduces execution time overheads.
Execution of the ITWS implemented using a combination of BPEL constructs and Java as BPEL extension takes the same time as execution of the ITWS implemented using BPEL constructs only

Execution of the ITWS implemented using Java as BPEL extension only takes the same time as execution of the ITWS implemented using BPEL constructs only

Execution of the ITWS implemented using Java as BPEL extension only takes the same time as execution of the ITWS implemented using a combination of BPEL constructs and Java as BPEL extension

Null hypothesis Alternative hypothesis Confidence interval p-value Null hypothesis rejected?

Execution of the ITWS implemented using a combination of BPEL constructs and Java as BPEL extension takes the same time as execution of the ITWS implemented using BPEL constructs only

Execution of the ITWS implemented using Java as BPEL extension only takes the same time as execution of the ITWS implemented using BPEL constructs only

Execution of the ITWS implemented using Java as BPEL extension only takes the same time as execution of the ITWS implemented using a combination of BPEL constructs and Java as BPEL extension

Table 8.5: 2-Sample Tests Results for ITWS ran with five WSs

<table>
<thead>
<tr>
<th>Null hypothesis</th>
<th>Alternative hypothesis</th>
<th>Confidence interval</th>
<th>p-value</th>
<th>Null hypothesis rejected?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution of the ITWS implemented using a combination of BPEL constructs and Java as BPEL extension takes the same time as execution of the ITWS implemented using BPEL constructs only</td>
<td>Execution of the ITWS implemented using a combination of BPEL constructs and Java as BPEL extension takes longer than execution of the ITWS implemented using BPEL constructs only</td>
<td>95%</td>
<td>2.09</td>
<td>No</td>
</tr>
<tr>
<td>Execution of the ITWS implemented using Java as BPEL extension only takes the same time as execution of the ITWS implemented using BPEL constructs only</td>
<td>Execution of the ITWS implemented using Java as BPEL extension only takes longer than execution of the ITWS implemented using BPEL constructs only</td>
<td>95%</td>
<td>0.000</td>
<td>Yes</td>
</tr>
<tr>
<td>Execution of the ITWS implemented using Java as BPEL extension only takes the same time as execution of the ITWS implemented using a combination of BPEL constructs and Java as BPEL extension</td>
<td>Execution of the ITWS implemented using Java as BPEL extension only takes longer than execution of the ITWS implemented using a combination of BPEL constructs and Java as BPEL extension</td>
<td>95%</td>
<td>0.000</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Figure 8.7: Dynamic Reconfiguration Time (ms) for ITWS Implemented using Combination of Java and BPEL Constructs

Figure 8.8: Dynamic Reconfiguration Time (ms) for ITWS Implemented using Java only
In addition to the previous evaluations, the reconfiguration times of ITWS implemented using Java as BPEL extension only and ITWS implemented using a combination of BPEL constructs and Java as BPEL extension are also compared (see Figures 8.7 and 8.8). For this examination, deliberate faults (two of the WSs developed in Section 7.1.1 are altered to return their response in unexpected format) are injected into these ITWSs forcing them to perform reconfiguration. The result shows 34.35% increase in the reconfiguration time in the case of ITWS implemented using only Java as BPEL extension approach.

As presented in this section, the answer to the research question five is yes, the use of Java as BPEL extension enables dynamic reconfiguration of ITWS.

And the answer to the central question underlying this work (Can a self-reconfigurable ITWS, implemented using N-version programming and diversity formed by composing OTSWSs that are selected through penetration testing, PCA, and CA processes mitigate XML-related security vulnerabilities?) is:

Yes, a composite WS implemented in BPEL can be converted to a self-reconfigurable ITWS using the proposed approaches, presented in this dissertation, to mitigate XML-related security vulnerabilities.
This chapter presents work related to this dissertation. Section 9.1 discusses related security standards, Intrusion Detection (ID) and Intrusion Prevention (IP) approaches. It also reviews a number of other Intrusion-Tolerant systems solutions. Section 9.2 presents other service selection solutions, including those based on PCA and CA approaches and compares them to the service selection framework of this dissertation. Finally, Section 9.3 discusses other works related to the dynamic reconfiguration and adaptation of composite WSs.

9.1 Intrusion Detection, Prevention and Tolerant Systems

This section reviews related security standards and a number of related ID/IP approaches specific to WSs as well as a number of related IT-WSs.

9.1.1 Related Security Standards

Various standards (including WS-Security, WS-Policy, WS-Trust, WS-Privacy, WS-Federation, WS-SecureConversation, WS-Authorization and etc.) have been created to be used as building blocks of the WSs to protect them against XML-related cyber-attacks [130]. However, they have limitations; such as their implementation might introduce some complexity and
that they may cause security vulnerabilities if not implemented properly (see Chapter 2 for more details).

9.1.2 Related ID/IP Approaches

Lindstrom [51] proposed to first validate XML documents, using format and syntax inspection and validation approaches, to conform to the XML and SOAP specifications then perform a deeper inspection of the content, looking for any policy violations (e.g., oversized documents). Loh et al. [131] proposed an ID/IP framework for WSs, which is a combination of syntax parsing on SOAP messages (to check the structure of XML for syntax errors), filtering policy to check and restrict the size of the SOAP message (to prevent Oversized Payload attacks) and XML schema validation (to counter SOAP flooding attacks). Similarly, Yee et al. [132] proposed an adaptive ID/IP framework for WSs, which consists of agents that act as sensors, data mining techniques such as clustering, association and sequential rule coupled with fuzzy logic to detect violations to the normal profile. The anomalies are then further analysed using fuzzy logic to determine genuine attacks to reduce false alarms. In the event of an attack, the action provider either blocks, rejects or terminates the activity. In [133] an ID framework is proposed that detects cyber-attacks on WSs using XML similarity classifiers. This framework consists of: (1) monitoring every request/response for identification of any special characters that may cause SQL/XML Injection attacks; (2) a validation scheme using WSDL definition of the WS with the aim of preventing XML DoS attacks; and (3) a normal dataset of requests (based on the normal behaviours of the previous traces) as a reference for comparing both the structure and the semantics of every input request. Gorbenko et al. [134] proposed a diversity-aware IP framework based on multi-level software diversity and dynamic software reconfiguration for deploying WSs in a cloud. In this framework, the vulnerability information from sources such as NVD and CVE are utilized at real-time to compute the vulnerability scores for the available diverse software infrastructures. The infrastructure with the less vulnerability score is then chosen and redeployed if a different infra-
structure is currently deployed. They do not consider replicated systems but choose the best single configuration from a pool of diverse options.

Generally, the ID/IP systems may effectively detect pre-defined attacks but have limitations in responding to continuously created novel attacks.

9.1.3 Related Intrusion-Tolerant Systems

A considerable volume of work in this area has focused on implementing Intrusion-Tolerant systems based on diverse redundant components [135]-[138]. The basic intuition is that design diversity reduces the possibility of common security vulnerabilities that are exploited by an attack. These systems are based on active replication techniques. There are proxy servers which interface with the external world mediating access to the actual servers. The inputs to the servers and the responses from the servers are passed through validity checks. Voting is performed on the responses from the servers and any disagreement acts as a trigger for the reconfiguration (e.g., bringing in a different server).

Kalkhoran et al. [139] proposed an ITWS that uses simple primary and backup scheme (both services have the same implementation, hence, the same security vulnerabilities). In this framework, the available operations are extracted from the WSDL of the WS, and then an XML schema is generated and hardened and used to validate the incoming messages. The objective of this ITWS is to prevent previously detected attacks from occurring continually on the system. For this purpose, the system’s activity patterns are utilized to detect misuses or abnormal behaviours. Following detection of the malicious requests, the containment module tries to extend these requests to attack patterns and add them to the attack patterns database. In the event of a successful intrusion, the compromised services are disabled and the reconfiguration manager checks whether the service level is satisfactory. Otherwise, the online server is restored to a clean state and the hot standby copy is promoted to the online server.
9.2 Service Selection

Existing approaches to service discovery and selection are mainly based on non-functional properties such as QoS, policies, trust, and reputation from client perspective [140]. In these approaches, the security property is defined as a set of high-level QoS attributes (e.g., confidentiality and integrity) usually claimed by the service providers with no supporting evidence. The service selection approach presented in [141] is based on both functional and non-functional requirements and contains a QoS certifier that checks the QoS claims made by the service providers. Maximilien and Singh [142] proposed a runtime service selection framework that utilizes agents (acting as proxies between the clients and the services) discovering services based on semantics and QoS policies. Quing et al. [140] present a survey of service selection approaches based on non-functional requirements. Yau and Yin [143] proposed a service selection framework based on QoS metrics integrated into a single satisfaction score. The service selection solutions presented in [144] is based on the client’s preferences and WS’s properties. It combines logic-based and optimization methods. Chaari et al. [145] proposed a service selection approach based on ontology reasoning and an extension to WS-Policy. The service selection framework presented in [146] is based on the trust and the reputation of the WSs. This approach evaluates the trustworthiness of the providers based on their reputation then provides a reputation-based service discovery methodology driven by clients’ QoS preferences. Anisetti et al. [147] proposed a service selection framework that selects the WSs based on their security certificates that best satisfy the client’s preference. Finally, Qi et al. [148] proposed service selection method based on weighted PCA. They have argued that the weighted PCA may reduce the number of QoS criteria simplifying the service selection process and eliminate the correlations between different QoS criteria increasing service selection accuracy.

The service selection framework presented in this dissertation differs from the above solutions as it is based on tested security vulnerabilities of the WSs (not just the service provider’s claim)
9.3 Reconfiguration/Adaptation

In standard BPEL processes, the interactions with other WSs are enabled through PortTypes, which are defined statically during the implementation of the business process. A much more controllable and hence flexible approach is to enable dynamic runtime lookup, selection, and binding to such business processes.

Ezenwoye et al. [149] presented RobustBPEL framework, which can generate an adaptable version of an existing BPEL process. The BP generated through this framework monitors the invocation of partner WSs and invokes a static proxy service in the event of a failure. This proxy looks for an equivalent service to replace the failed one. In this approach, the information about the equivalent services is hardcoded at proxy generation time. Later they proposed RobustBPEL2 [150], which is uses dynamic proxies that enables the runtime discovery of equivalent services. Furthermore, RobustBPEL2 adds self-optimizing capabilities to existing BPEL processes. While their work is similar to the reconfiguration approaches presented in this dissertation in respect to their aim to improve reliability in the context of BPEL and WSs, they are using a proxy based approach, which requires the interface of the equivalent services to match the proxy service, to monitor process execution and improve process performance, whereas the approaches presented in this dissertation leverage Java as BPEL extension, which enables invocation of any equivalent service, to improve dependability of the BP. Hence, the approaches presented in this dissertation enable better self-reconfiguration. RobustBPEL2 uses UDDI to discover alternative services in the event of a failure. However, it does not incorporate selection criteria when multiple services are found, while this work chooses the most secure available group of WSs from the database.

In [151], Baresi et al. proposed a framework enabling self-healing capabilities for BPEL processes. Their framework is based on implementation of the Dynamo [152] framework (a supervision framework), which consists of an AOP-extended version of the ActiveBPEL and is built using the JBoss Rule Engine. Similarly, they employ BPEL extensions however, of AOP type
not Java. Their solution does not explicitly address the problem of selecting alternative services whereas; the solution presented in this work provides a viable way to select alternative services.

AO4BPEL [153] is an extension to BPEL4WS. It is an aspect-oriented approach to WSs composition, which provides aspect-oriented modularity mechanism. Aspects are defined in XML documents and in the case of AO4BPEL they are BPEL activities that implement cross-cutting concerns (e.g., security) or workflow changes. To employ the aspect, it must be registered with the BPEL’s execution engine. This registration can be done during the runtime of the BP. However, it requires information such as the WSDL and port address of the service, as well as the partnerLinkTypes (necessary information for establishing communication with the external WSs) that are to be used by the aspect. Then, AO4BPEL can support dynamically changing the deployed process through activating/deactivating aspects. Whereas, one of the advantages of the self-reconfiguration approaches demonstrated in this work is that the BP collects the necessary information for establishing communication with the external WSs during its execution time. This feature gives the backend of the BP the freedom of updating the services’ records in the DB (including adding and deleting services), which enables the ITWS to stay up to date.

In [154], Kongdenfha et al. proposed an aspect-oriented framework enabling service adaptation. Their approach uses aspect-based templates to automate the task of handling interface mismatches (including protocol mismatches). Whereas, the approaches presented in this work utilize self-reconfiguration to support Intrusion-Tolerance.
This dissertation is concerned with improving the dependability of Web Services (WSs) especially when Off-The-Shelf Web Services (OTSWSs) are employed. It introduced a novel framework to increase dependability by constructing Intrusion-Tolerant Web Services (ITWSs) in which *N-version programming* and *diversity*, formed by composing OTSWSs, are used. It also demonstrated implementation of self-reconfiguration ITWS using a combination of Business Process Engineering Language (BPEL) constructs and Java as BPEL extension approach and using only Java as BPEL extension approach.

10.1 Summary

Chapter 1 introduced this work and its context (using an exemplary WS). It also briefly discussed the motivation for this work and presented the central research question it was going to answer as well as its contributions and distinctions. Finally, it briefly outlined the other chapters.

Chapter 2 presented the overview of WSs’ architecture and introduced its main technologies (SOAP, WSDL, and UDDI). It then explained that WSs are at risk of security vulnerabilities related to their specific implementation technologies (e.g., XML) as well as those, of their underlying platforms (e.g., operating systems) and web Applications (e.g., vulnerability to SQL Injection attacks). Afterward, it introduced a number of existing coun-
termmeasures against attacks targeting WSs’ XML-related vulnerabilities followed by a list of their limitations and argued that the issue gets more challenging when OTSWSs are employed as they are ready-made black boxes of unknown quality and their security is out of the control of their client thus, tolerating their security vulnerabilities through a reconfigurable ITWS is a more appropriate approach. It then briefly introduced dependability approaches and explained that ITWS could be achieved using dependability techniques which, is the approach adopted in this work and is achieved through the integration of WSs’ composability, Principal Component Analysis (PCA), Cluster Analysis (CA) and penetration testing. After that, it briefly introduced each of these concepts and explained the motivation for their adoption in this work.

Chapter 3 presented the architecture for this framework along with its objectives and the assumptions made. It then explained the role of each of its components and the interactions among them.

Chapter 4 introduced the penetration testing tool, utilized in the case studies presenting the proposed service selection framework. It then demonstrated the feasibility of ITWS implementation based on penetration test results of the candidate WSs.

Chapter 5 showed that BPEL could affect the XML-related security vulnerabilities of the candidate (for ITWS implementation) WSs and argued that these effects should be considered in service selection process.

Chapter 6 demonstrated, how penetration test results of the candidate WSs, PCA, and CA could be used to group WSs based on their XML-related security vulnerabilities and how these groups could be sorted using further penetration testing.

Chapter 7 demonstrated the implementation of self-reconfigurable WS using a combination of BPEL constructs and Java as BPEL 2.0 extension approach and utilizing only Java as BPEL extension approach through two case studies.
Chapter 8 evaluated this work using the experimental results collected from the case studies that demonstrated different dimensions of the work presented in this dissertation. It discussed the advantages, extensibility, and limitations of the proposed framework. Finally, it answered the underlying central research question.

Chapter 9 discussed other related approaches.

10.2 Future Work

During this work, the author identified extensions of the current dissertation and future directions of this research as outlined below.

- Diversity could also be applied to penetration testing tools and tests.
- XML hardening could be applied to some of the utilized WSs to make them more secure aiming to increase diversity among the WSs.
- Diversity could also be applied at the composition level since the BPEL itself could be a single point of failure.
- Automation could be applied to service discovery stage, which is currently the responsibility of the system’s administrator.
- Recall that WS-Attacker’s plugins also simulate concurrent legitimate users. These experiments could be repeated while this feature is completely switched off (to diminish the fluctuations in the penetration testing results).
- The proposed service selection framework could be further evaluated using WS-groups formed based on remaining combinations of the WSs (identified by PCA and CA as non-optimal WS-groups according to their overall XML-related security vulnerabilities).

10.3 Conclusions

This dissertation makes the following main conclusions:

An ITWS formed based on penetration test results of candidate WSs, is a feasible approach to mitigate XML-related security vulnerabilities.
PCA and CA improve the process of security-aware service selection based on penetration testing results of candidate WSs.

PCA is an effective pre-processing step to CA in the proposed service selection framework.

An ITWS in which N-version programming and diversity, formed by composing SOAP-OTSWSs (elected through PCA and CA analysis on their penetration testing results) is used, is a feasible approach to mitigate XML-related security vulnerabilities.

Java as BPEL extension enables to implement self-reconfigurable ITWS in return for the cost of longer execution time.

And the answer to the central question underlying this work (Can a self-reconfigurable ITWS, implemented using N-version programming and diversity formed by composing OTSWSs that are selected through penetration testing, PCA, and CA processes mitigate XML-related security vulnerabilities?) is:

*Yes, a composite WS implemented in BPEL can be converted to a self-reconfigurable ITWS using the proposed approaches, presented in this dissertation, to mitigate XML-related security vulnerabilities.*
References


Appendix A: WSDL of Axis1-4 WS running on apache-tomcat-6.0.18 server

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://Axis_Tom_6"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://Axis_Tom_6"
xmlns:intf="http://Axis_Tom_6"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
<wsdl:types>
    <schema elementFormDefault="qualified" targetNamespace="http://Axis_Tom_6"
xmlns="http://www.w3.org/2001/XMLSchema">
        <element name="factorial">
            <complexType>
                <sequence>
                    <element name="n" type="xsd:int"/>
                </sequence>
            </complexType>
        </element>

        <element name="factorialResponse">
            <complexType>
                <sequence>
                    <element name="factorialReturn" type="xsd:int"/>
                </sequence>
            </complexType>
        </element>

        <element name="addIntegers">
            <complexType>
                <sequence>
                    <element name="firstNum" type="xsd:int"/>
                    <element name="secondNum" type="xsd:int"/>
                </sequence>
            </complexType>
        </element>

        <element name="addIntegersResponse">
            <complexType>
                <sequence>
                </sequence>
            </complexType>
        </element>
    </schema>
</wsdl:types>
</wsdl:definitions>
```
"addIntegersReturn" type="xsd:int"/>

</sequence>
</complexType>
</schema>
</wsdl:types>
<wsdl:message name="factorialResponse">
<wsdl:part element="impl:factorialResponse" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:message name="factorialRequest">
<wsdl:part element="impl:factorial" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:message name="addIntegersRequest">
<wsdl:part element="impl:addIntegers" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:message name="addIntegersResponse">
<wsdl:part element="impl:addIntegersResponse" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:portType name="Axis_Tom_6_sum">
<wsdl:operation name="factorial">
<wsdl:input message="impl:factorialRequest" name="factorialRequest">
</wsdl:input>
<wsdl:output message="impl:factorialResponse" name="factorialResponse">
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="addIntegers">
<wsdl:input message="impl:addIntegersRequest" name="addIntegersRequest">
</wsdl:input>
<wsdl:output message="impl:addIntegersResponse" name="addIntegersResponse">
</wsdl:output>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="Axis_Tom_6_sumSoapBinding" type="impl:Axis_Tom_6_sum">
<wsdl:operation name="factorial"/>
<wsdl:definitions>

<wsdl:operation soapAction=""/>
<wsdl:input name="factorialRequest">
  <wsdlsoap:body use="literal"/>
</wsdl:input>
<wsdl:output name="factorialResponse">
  <wsdlsoap:body use="literal"/>
</wsdl:output>
</wsdl:operation>

<wsdl:operation name="addIntegers">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="addIntegersRequest">
    <wsdlsoap:body use="literal"/>
  </wsdl:input>
  <wsdl:output name="addIntegersResponse">
    <wsdlsoap:body use="literal"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:binding>
  <wsdl:servicename="Axis_Tom_6_sumService">
    <wsdl:port binding="impl:Axis_Tom_6_sumSoapBinding" name="Axis_Tom_6_sum">
      <wsdlsoap:address location="http://localhost:8888/Axis_Tom_6/services/Axis_Tom_6_sum"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:binding>

</wsdl:definitions>
Appendix B: WSDL of Axis2-1.5.1 WS running on apache-tomcat-6.0.18 server

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:ns="http://Axis2_1_5_1_Tom_6"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:ns1="http://org.apache.axis2/xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" target-
Namespace="http://Axis2_1_5_1_Tom_6">
  <wsdl:documentation>
    Please Type your service description here
  </wsdl:documentation>
  <wsdl:types>
    <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" target-
Namespace="http://Axis2_1_5_1_Tom_6">
      <xs:element name="addIntegers">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="firstNum" type="xs:int"/>
            <xs:element minOccurs="0" name="secondNum" type="xs:int"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="addIntegersResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="return" type="xs:int"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="factorial">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="n" type="xs:int"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>
</wsdl:definitions>
```
<xs:element name="factorialResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="return" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

<wsdl:types>
  <wsdl:message name="factorialRequest">
    <wsdl:part name="parameters" element="ns:factorial"/>
  </wsdl:message>
  <wsdl:message name="factorialResponse">
    <wsdl:part name="parameters" element="ns:factorialResponse"/>
  </wsdl:message>
  <wsdl:message name="addIntegersRequest">
    <wsdl:part name="parameters" element="ns:addIntegers"/>
  </wsdl:message>
  <wsdl:message name="addIntegersResponse">
    <wsdl:part name="parameters" element="ns:addIntegersResponse"/>
  </wsdl:message>
</wsdl:types>

<wsdl:portType name="Axis2_1_5_1_Tom_6_sumPortType">
  <wsdl:operation name="factorial">
    <wsdl:input message="ns:factorialRequest" wsaw:Action="urn:factorial"/>
    <wsdl:output message="ns:factorialResponse" wsaw:Action="urn:factorialResponse"/>
  </wsdl:operation>
  <wsdl:operation name="addIntegers">
    <wsdl:input message="ns:addIntegersRequest" wsaw:Action="urn:addIntegers"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Axis2_1_5_1_Tom_6_sumSoap11Binding" type="ns:Axis2_1_5_1_Tom_6_sumPortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <wsdl:operation name="factorial">
    <soap:operation soapAction="urn:factorial" style="document"/>
    <wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
    <soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>

<wsdl:operation name="addIntegers">
    <soap:operation soapAction="urn:addIntegers" style="document"/>
    <wsdl:input>
        <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output>
        <soap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
</wsdl:binding>

<wsdl:binding name="Axis2_1_5_1_Tom_6_sumSoap12Binding"
type="ns:Axis2_1_5_1_Tom_6_sumPortType">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="factorial">
        <soap12:operation soapAction="urn:factorial" style="document"/>
        <wsdl:input>
            <soap12:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>

<wsdl:binding name="Axis2_1_5_1_Tom_6_sumSoap12Binding"
type="ns:Axis2_1_5_1_Tom_6_sumPortType">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="addIntegers">
        <soap12:operation soapAction="urn:addIntegers" style="document"/>
        <wsdl:input>
            <soap12:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>

<wsdl:binding name="Axis2_1_5_1_Tom_6_sumSoap12Binding"
type="ns:Axis2_1_5_1_Tom_6_sumPortType">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="addIntegers">
        <soap12:operation soapAction="urn:addIntegers" style="document"/>
        <wsdl:input>
            <soap12:body use="literal"/>
        </wsdl:input>
        <wsdl:output>
            <soap12:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<http:binding verb="POST"/>
<wsdl:operation name="factorial">
  <http:operation location="Axis2_1_5_1_Tom_6_sum/factorial"/>
  <wsdl:input>
    <mime:content type="text/xml" part="factorial"/>
  </wsdl:input>
  <wsdl:output>
    <mime:content type="text/xml" part="factorial"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="addIntegers">
  <http:operation location="Axis2_1_5_1_Tom_6_sum/addIntegers"/>
  <wsdl:input>
    <mime:content type="text/xml" part="addIntegers"/>
  </wsdl:input>
  <wsdl:output>
    <mime:content type="text/xml" part="addIntegers"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:service name="Axis2_1_5_1_Tom_6_sum">
  <wsdl:port name="Axis2_1_5_1_Tom_6_sumHttpSoap11Endpoint" binding="ns:Axis2_1_5_1_Tom_6_sumSoap11Binding">
    <soap:address location="http://localhost:8888/Axis2_1_5_1_Tom_6/services/Axis2_1_5_1_Tom_6_sum.Axis2_1_5_1_Tom_6_sumHttpSoap11Endpoint"/>
  </wsdl:port>
  <wsdl:port name="Axis2_1_5_1_Tom_6_sumHttpSoap12Endpoint" binding="ns:Axis2_1_5_1_Tom_6_sumSoap12Binding">
    <soap12:address location="http://localhost:8888/Axis2_1_5_1_Tom_6/services/Axis2_1_5_1_Tom_6_sum.Axis2_1_5_1_Tom_6_sumHttpSoap12Endpoint"/>
  </wsdl:port>
  <wsdl:port name="Axis2_1_5_1_Tom_6_sumHttpEndpoint" binding="ns:Axis2_1_5_1_Tom_6_sumHttpBinding">
    <http:address location="http://localhost:8888/Axis2_1_5_1_Tom_6/services/Axis2_1_5_1_Tom_6_sum.Axis2_1_5_1_Tom_6_sumHttpEndpoint"/>
  </wsdl:port>
</wsdl:service>
Appendix C: WSDL of Axis2-1.6.1 WS running on apache-tomcat-6.0.18 server

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wSDL:definitions
xmlns:wSDL="http://schemas.xmlsoap.org/wsd/" xmlns:ns="http://Axis2_1_6_1_Tom_6"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsd/"
xmlns:mime="http://schemas.xmlsoap.org/wsd/mime/"
xmlns:http="http://schemas.xmlsoap.org/wsd/http/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsd/soap12/"
xmlns:ns1="http://org.apache.axis2/xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsd/soap/" target-
Namespace="http://Axis2_1_6_1_Tom_6">
    <wSDL:documentation>
        Please Type your service description here
    </wSDL:documentation>
    <wSDL:types>
        <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" target-
Namespace="http://Axis2_1_6_1_Tom_6">
        <xs:element name="addIntegers">
            <xs:complexType>
                <xs:sequence>
                    <xs:element minOccurs="0" name="firstNum" type="xs:int"/>
                    <xs:element minOccurs="0" name="secondNum" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="addIntegersResponse">
            <xs:complexType>
                <xs:sequence>
                    <xs:element minOccurs="0" name="return" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="factorial">
            <xs:complexType>
                <xs:sequence>
                    <xs:element minOccurs="0" name="n" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:schema>
</wSDL:types>
</wSDL:definitions>
```
<xs:complexType>
  <xs:sequence>
    <xs:element minOccurs="0" name="return" type="xs:int"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

<wsdl:types>
  <wsdl:message name="factorialRequest">
    <wsdl:part name="parameters" element="ns:factorial"/>
  </wsdl:message>
  <wsdl:message name="factorialResponse">
    <wsdl:part name="parameters" element="ns:factorialResponse"/>
  </wsdl:message>
  <wsdl:message name="addIntegersRequest">
    <wsdl:part name="parameters" element="ns:addIntegers"/>
  </wsdl:message>
  <wsdl:message name="addIntegersResponse">
    <wsdl:part name="parameters" element="ns:addIntegersResponse"/>
  </wsdl:message>
  <wsdl:portType name="Axis2_1_6_1_Tom_6_sumPortType">
    <wsdl:operation name="factorial">
      <wsdl:input message="ns:factorialRequest" wsaw:Action="urn:factorial"/>
      <wsdl:output message="ns:factorialResponse" wsaw:Action="urn:factorialResponse"/>
    </wsdl:operation>
    <wsdl:operation name="addIntegers">
      <wsdl:input message="ns:addIntegersRequest" wsaw:Action="urn:addIntegers"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="Axis2_1_6_1_Tom_6_sumSoap11Binding" type="ns:Axis2_1_6_1_Tom_6_sumPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="factorial">
      <soap:operation soapAction="urn:factorial" style="document"/>
      <wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="addIntegers">
<soap:operation soapAction="urn:addIntegers" style="document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:binding name="Axis2_1_6_1_Tom_6_sumSoap12Binding" type="ns:Axis2_1_6_1_Tom_6_sumPortType">
<soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
<wsdl:operation name="factorial">
<soap12:operation soapAction="urn:factorial" style="document"/>
<wsdl:input>
<soap12:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap12:body use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="addIntegers">
<soap12:operation soapAction="urn:addIntegers" style="document"/>
<wsdl:input>
<soap12:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap12:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:binding name="Axis2_1_6_1_Tom_6_sumHttpBinding" type="ns:Axis2_1_6_1_Tom_6_sumPortType">
<http:binding verb="POST"/>
<wsdl:operation name="factorial">
    <http:operation location="factorial"/>
    <wsdl:input>
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:input>
    <wsdl:output>
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:output>
</wsdl:operation>

<wsdl:operation name="addIntegers">
    <http:operation location="addIntegers"/>
    <wsdl:input>
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:input>
    <wsdl:output>
        <mime:content type="text/xml" part="parameters"/>
    </wsdl:output>
</wsdl:operation>

<wsdl:service name="Axis2_1_6_1_Tom_6_sum">
    <wsdl:port name="Axis2_1_6_1_Tom_6_sumHttpSoap11Endpoint" binding="ns:Axis2_1_6_1_Tom_6_sumHttpSoap11Binding">
        <soap:address location="http://localhost:8888/Axis2_1_6_1_Tom_6/services/Axis2_1_6_1_Tom_6_sum.Axis2_1_6_1_Tom_6_sumHttpSoap11Endpoint"/>
    </wsdl:port>
    <wsdl:port name="Axis2_1_6_1_Tom_6_sumHttpSoap12Endpoint" binding="ns:Axis2_1_6_1_Tom_6_sumHttpSoap12Binding">
        <soap12:address location="http://localhost:8888/Axis2_1_6_1_Tom_6/services/Axis2_1_6_1_Tom_6_sum.Axis2_1_6_1_Tom_6_sumHttpSoap12Endpoint"/>
    </wsdl:port>
    <wsdl:port name="Axis2_1_6_1_Tom_6_sumHttpEndpoint" binding="ns:Axis2_1_6_1_Tom_6_sumHttpEndpoint">
        <http:address location="http://localhost:8888/Axis2_1_6_1_Tom_6/services/Axis2_1_6_1_Tom_6_sum.Axis2_1_6_1_Tom_6_sumHttpEndpoint"/>
    </wsdl:port>
</wsdl:service>
Appendix D: WSDL of CXF-2.5.11 WS running on apache-tomcat-7.0.72 server

```xml
<?xml version='1.0' encoding='UTF-8'?><wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://CXF_2__5_11_Tom_7/">
    <wsdl:types>
        <schema xmlns:tns="http://CXF_2__5_11_Tom_7/">
            <import namespace="http://CXF_2__5_11_Tom_7/"
                schemaLocation="cxf_2__5_11_tom_7_sum_schema1.xsd"/>
        </schema>
    </wsdl:types>
    <wsdl:message name="addIntegers">
        <wsdl:part element="tns:addIntegers" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="factorialResponse">
        <wsdl:part element="tns:factorialResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="addIntegersResponse">
        <wsdl:part element="tns:addIntegersResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="factorial">
        <wsdl:part element="tns:factorial" name="parameters"/>
    </wsdl:message>
    <wsdl:portType name="CXF_2__5_11_Tom_7_sum">
        <wsdl:operation name="addIntegers">
            <wsdl:input message="tns:addIntegers" name="addIntegers"/>
            <wsdl:output message="tns:addIntegersResponse" name="addIntegersResponse"/>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```
<wsdl:operation name="factorial">
<wSDL:input message="tns:factorial" name="factorial">
</wsdl:input>
<wSDL:output message="tns:factorialResponse" name="factorialResponse">
</wsdl:output>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="CXF_2__5_11_Tom_7_sumServiceSoapBinding" type="tns:CXF_2__5_11_Tom_7_sum">
<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="addIntegers">
<soap:operation soapAction="" style="document"/>
<wsdl:input name="addIntegers">
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output name="addIntegersResponse">
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="factorial">
<soap:operation soapAction="" style="document"/>
<wsdl:input name="factorial">
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output name="factorialResponse">
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="CXF_2__5_11_Tom_7_sumService">
<wsdl:port binding="tns:CXF_2__5_11_Tom_7_sumSoapBinding" name="CXF_2__5_11_Tom_7_sumPort">
<soap:address location="http://localhost:8889/CXF_2__5_11_Tom_7/services/CXF_2__5_11_Tom_7_sumPort"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
Appendix E: WSDL of CXF-2.3.10 WS running on apache-tomcat-6.0.18 server

```xml
<?xml version='1.0' encoding='UTF-8'?><wsdl:definitions
name="CXF_2_3_10_Tom_6_sumService" target-
Namespace="http://CXF_2_3_10_Tom_6/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<import namespace="http://CXF_2_3_10_Tom_6/" schemaLoca-
tion="cxf_2_3_10_tom_6_sum_schema1.xsd"/>
</schema>
</wsdl:types>
<wsdl:message name="factorial">
<wsdl:part element="tns:factorial" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:message name="addIntegersResponse">
<wsdl:part element="tns:addIntegersResponse" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:message name="factorialResponse">
<wsdl:part element="tns:factorialResponse" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:message name="addIntegers">
<wsdl:part element="tns:addIntegers" name="parameters">
</wsdl:part>
</wsdl:message>
<wsdl:portType name="CXF_2_3_10_Tom_6_sum">
<wsdl:operation name="factorial">
<wsdl:input message="tns:factorial" name="factorial">
</wsdl:input>
<wsdl:output message="tns:factorialResponse" name="factorialResponse">
</wsdl:output>
```
<wsdl:operation name="addIntegers">
    <wsdl:input message="tns:addIntegers" name="addIntegers"/>
    <wsdl:output message="tns:addIntegersResponse" name="addIntegersResponse"/>
</wsdl:operation>

<wsdl:operation name="factorial">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="factorial">
        <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="factorialResponse">
        <soap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>

<wsdl:operation name="addIntegers">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="addIntegers">
        <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="addIntegersResponse">
        <soap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>

<wsdl:binding name="CXF_2_3_10_Tom_6_sumServiceSoapBinding" type="tns:CXF_2_3_10_Tom_6_sum">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="factorial">
        <soap:operation soapAction="" style="document"/>
        <wsdl:input name="factorial">
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="factorialResponse">
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="addIntegers">
        <soap:operation soapAction="" style="document"/>
        <wsdl:input name="addIntegers">
            <soap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="addIntegersResponse">
            <soap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="CXF_2_3_10_Tom_6_sumService">
    <wsdl:port binding="tns:CXF_2_3_10_Tom_6_sumServiceSoapBinding" name="CXF_2_3_10_Tom_6_sumPort">
        <soap:address location="http://localhost:8888/CXF_2_3_10_Tom_6/services/CXF_2_3_10_Tom_6_sumPort"/>
    </wsdl:port>
</wsdl:service>
<wsdl:definitions>
Appendix F: WSDL of CXF-2.6.3 WS running on apache-tomcat-7.0.72 server

```xml
<?xml version='1.0' encoding='UTF-8'?><wsdl:definitions
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://CXF_2_6_3_Tom_7/
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  name="CXF_2_6_3_Tom_7_sumService" targetNamespace="http://CXF_2_6_3_Tom_7/">
  <wsdl:types>
    <schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:tns="http://CXF_2_6_3_Tom_7/
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://CXF_2_6_3_Tom_7/"
        schemaLocation="cxf_2_6_3_tom_7_sum_schema1.xsd"/>
    </schema>
  </wsdl:types>
  <wsdl:message name="factorial">
    <wsdl:part element="tns:factorial" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="factorialResponse">
    <wsdl:part element="tns:factorialResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="addIntegers">
    <wsdl:part element="tns:addIntegers" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="addIntegersResponse">
    <wsdl:part element="tns:addIntegersResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="CXF_2_6_3_Tom_7_sum">
    <wsdl:operation name="factorial">
      <wsdl:input message="tns:factorial" name="factorial"/>
      <wsdl:output message="tns:factorialResponse" name="factorialResponse"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```
<wsdl:operation name="addIntegers">
  <wsdl:input message="tns:addIntegers" name="addIntegers"/>
</wsdl:input>
<wsdl:output message="tns:addIntegersResponse" name="addIntegersResponse"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="CXF_2_6_3_Tom_7_sumServiceSoapBinding" type="tns:CXF_2_6_3_Tom_7_sum">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="factorial">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="factorial">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="factorialResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="addIntegers">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="addIntegers">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="addIntegersResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="CXF_2_6_3_Tom_7_sumService">
  <wsdl:port binding="tns:CXF_2_6_3_Tom_7_sumServiceSoapBinding" name="CXF_2_6_3_Tom_7_sumPort">
    <soap:address location="http://localhost:8889/CXF_2_6_3_Tom_7/services/CXF_2_6_3_Tom_7_sumPort"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
Appendix G: Java Class for Communication between ITWS and Database

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Collections;

public class DBClass {
    public static String create(int group, int priority){
        String result = "";
        try{
            Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@//localhost:1521/xe", "sys as sysdba", "Passwor123");
            Statement stmt =
                    conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ _ONLY);
            Statement stmt2 =
                    conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ _ONLY);
            ResultSet rset = stmt.executeQuery("SELECT ENDPOINT,TARGETNAMESPACE,OPERATION,INPUT FROM HR.WEBSERVICES " +
                    "WHERE SERVICEGROUP =" + group + "AND PRIORITY = " + priority);
            int i = 0;
            ArrayList<String> ENDPOINTS = new ArrayList<String>();
            ArrayList<String> TARGETNAMESPACES = new ArrayList<String>();
            ArrayList<String> OPERATIONS = new ArrayList<String>();
            ArrayList<String> INPUTS = new ArrayList<String>();
            while (rset.next()) {
                ENDPOINTS.add(i, rset.getString("ENDPOINT"));
                TARGETNAMESPACES.add(i, rset.getString("TARGETNAMESPACE"));
                OPERATIONS.add(i, rset.getString("OPERATION"));
                INPUTS.add(i, rset.getString("INPUT"));
                i++;
            }
            for (int j = 0; j < ENDPOINTS.size(); j++) {
                result += ENDPOINTS.get(j) + "," + TARGETNAMESPACES.get(j) + "," + OPERATIONS.get(j)
            }
        }
    }
```
ResultSet rset2 = stmt2.executeQuery("SELECT PRIORITY FROM HR.WEBSERVICES");
int x = 0;
ArrayList<Integer> PRIORITY = new ArrayList<Integer>();

while (rset2.next()) {
    PRIORITY.add(x, Integer.parseInt(rset2.getString("PRIORITY")));
    i++;
}
result += String.valueOf(Collections.max(PRIORITY));
conn.close();
} catch (Exception ex) {
ex.printStackTrace();
}
return result;
}
Appendix H: Java Class for Dynamic WS Invocation Using RPC Library

```java
import java.util.Iterator;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.MimeHeaders;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;

public class Client {

    //Starting point for the SAAJ - SOAP Client Testing
    public static String create(String serviceInfo) {
        String result = "";
        String[] serviceInformation = serviceInfo.split("\,");  
        try {
            // Create SOAP Connection
            SOAPConnectionFactory soapConnectionFactory = SOAPConnectionFactory.newInstance();
            SOAPConnection soapConnection = soapConnectionFactory.createConnection();
            // Send SOAP Message to SOAP Server
            String url = serviceInformation[0];
            SOAPMessage soapResponse = soapConnection.call(createSOAPRequest(serviceInformation[1], serviceInformation[2], serviceInformation[3], serviceInformation[4]), url);
            // Process the SOAP Response
            result = printSOAPResponse(soapResponse);
            soapConnection.close();
        } catch (Exception e) {
            System.err.println("Error occurred while sending SOAP Request to Server");
        }
    }

    private static SOAPMessage createSOAPRequest(String serviceName, String serviceVersion, String operationName, String input) {
        // SOAP message creation logic
        return null;  
    }

    private static String printSOAPResponse(SOAPMessage response) {
        // SOAP response processing logic
        return null;
    }
}
```
private static String printSOAPResponse(SOAPMessage soapResponse) throws Exception {
    String result = "";
    SOAPBody responseBody = soapResponse.getSOAPBody();
    Iterator it1 = responseBody.getChildElements();
    while (it1.hasNext()) {
        SOAPBodyElement bodyEl = (SOAPBodyElement)it1.next();
        Iterator it2 = bodyEl.getChildElements();
        while (it2.hasNext()) {
            SOAPElement child2 = (SOAPElement)it2.next();
            result = child2.getValue();
        }
    }
    return result;
}

private static SOAPMessage createSOAPRequest(String namespace, String operation,
        String input1, String input2) throws Exception {
    MessageFactory messageFactory = MessageFactory.newInstance();
    SOAPMessage soapMessage = messageFactory.createMessage();
    SOAPPart soapPart = soapMessage.getSOAPPart();
    String serverURI = namespace;
    // SOAP Envelope
    SOAPEnvelope envelope = soapPart.getEnvelope();
    envelope.addNamespaceDeclaration("example", serverURI);
    // SOAP Body
    SOAPBody soapBody = envelope.getBody();
    SOAPElement soapBodyElem = soapBody.addChildElement(operation, "example");
    SOAPElement soapBodyElem1 = soapBodyElem.addChildElement(input1, "example");
    soapBodyElem1.addTextNode("10");
    SOAPElement soapBodyElem2 = soapBodyElem.addChildElement(input2, "example");
    soapBodyElem2.addTextNode("12");
    MimeHeaders headers = soapMessage.getMimeHeaders();
    headers.addHeader("SOAPAction", serverURI + operation);
soapMessage.saveChanges();
/* Print the request message */
System.out.print("Request SOAP Message = ");
soapMessage.writeTo(System.out);
System.out.println();
return soapMessage;
}
Appendix I: Java Class for Dynamic WS Invocation through RPC and Java Multi-Threading Libraries

```java
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

import javax.xml.soap.MessageFactory;
import javax.xml.soap.MimeHeaders;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;

public class Client2 {

    public static String create(String input) {
        String result = "";
        ExecutorService executorService = Executors.newSingleThreadExecutor();
        Set<Callable<String>> callables = new HashSet<Callable<String>>();
        String[] serviceInfo = input.split("\n");
        String[] serviceInfo = input.split("\n");
        for (int i = 0; i < (serviceInfo.length)-1; i++) {
            final int counter = i;
            callables.add(new Callable<String>() {
                public String call() throws Exception {
                    return serviceClient(serviceInfo[counter]);
                }
            });
        }

        return result;
    }
}
```

try {
    List<Future<String>> futures = executorService.invokeAll(callables);
    for(Future<String> future : futures){
        System.out.println("future.get = " + future.get());
        result += future.get() + ",";
    }
    executorService.shutdown();
} catch (Exception e) {
    e.printStackTrace();
}
return result;
}

public static String serviceClient(String input) {
    String result = "";
    String[] serviceInfo = input.split("\\,");
    try {
        // Create SOAP Connection
        SOAPConnectionFactory soapConnectionFactory = SOAPConnectionFactory.newInstance();
        SOAPConnection soapConnection = soapConnectionFactory.createConnection();
        // Send SOAP Message to SOAP Server
        String url = serviceInfo[0];
        SOAPMessage soapResponse = soapConnection.call(createSOAPRequest(serviceInfo[1], serviceInfo[2],
                                                                       serviceInfo[3], serviceInfo[4]), url);
        result = printSOAPResponse(soapResponse);
        soapConnection.close();
    } catch (Exception e) {
        System.err.println("Error occurred while sending SOAP Request to Server");
        e.printStackTrace();
    }
    return result;
}

private static SOAPMessage createSOAPRequest(String namespace, String operation, String input1, String input2) throws Exception {
    MessageFactory messageFactory = MessageFactory.newInstance();
    SOAPMessage soapMessage = messageFactory.createMessage();
SOAPPart soapPart = soapMessage.getSOAPPart();
String serverURI = namespace;
SOAPEnvelope envelope = soapPart.getEnvelope();
envelope.addNamespaceDeclaration("example", serverURI);
SOAPBody soapBody = envelope.getBody();
SOAPElement soapBodyElem = soapBody.addChildElement(operation, "example");
SOAPElement soapBodyElem1 = soapBodyElem.addChildElement(input1, "example");
soapBodyElem1.addTextNode("10");
SOAPElement soapBodyElem2 = soapBodyElem.addChildElement(input2, "example");
soapBodyElem2.addTextNode("12");
MimeHeaders headers = soapMessage.getMimeHeaders();
headers.addHeader("SOAPAction", serverURI + operation);
soapMessage.saveChanges();
System.out.print("Request SOAP Message = ");
soapMessage.writeTo(System.out);
System.out.println();
return soapMessage;
}
// Method used to print the SOAP Response
private static String printSOAPResponse(SOAPMessage soapResponse) throws Exception {
    String result = "";
    SOAPBody responseBody = soapResponse.getSOAPBody();
    Iterator it1 = responseBody.getChildElements();
    while (it1.hasNext()) {
        SOAPBodyElement bodyEl = (SOAPBodyElement)it1.next();
        Iterator it2 = bodyEl.getChildElements();
        while (it2.hasNext()) {
            SOAPElement child2 = (SOAPElement)it2.next();
            result = child2.getValue();
        }
    }
    return result;
}
Appendix J: Remaining Java Codes for Chapter 7

Code J.1 Java as BPEL Extension for Executing Database Query Class

```java
String priority1 = (String) getVariableData("Priority");
String group1 = (String) getVariableData("Service-Group");
int priority = Integer.parseInt(priority1);
int group = Integer.parseInt(group1);

String result = DBClass.create(group, priority);
setVariableData("DB-Result", result);

String[] counter = result.split("\\\[");
setVariableData("Loop-Cycles", Integer.parseInt(counter[1]));
String[] result2 = counter[0].split("\\\]");
setVariableData("Max-Priority-From-DB", result2[(result2.length) - 1]);
```

Code J.2 Java as BPEL Extension for Executing Dynamic Service Invocation Class

```java
String input = (String) getVariableData("DB-Result");
int counter = (int) getVariableData("ForEach1Counter");
String[] result = input.split("\\\[");
setVariableData("Invocation-Result", getVariableData("Invocation-Result") + Client.create(result[counter]) + ",");
```

Code J.3 Java Code for Performing Majority Voting

```java
String value = (String) getVariableData("Invocation-Result");
String[] values = value.split(";");
int maxValue = 0, maxCount = 0;
for (int i = 0; i < values.length; i++) {
    int count = 0;
    for (int j = 0; j < values.length; ++j) {
        if (values[j].equals(values[i])) {
            ++count;
        }
        if (count > maxCount) {
            maxCount = count;
            maxValue = values[i];
        }
    }
}
setVariableData("output", maxValue);
```
Code J.4 Java Code for Performing Fault-Tolerance

```java
String priority1 = (String) getVariableData("Priority");
int priority = Integer.parseInt(priority1);

String maximum1 = (String) getVariableData("Max-Priority-From-DB");
int maximum = Integer.parseInt(maximum1);

if(priority <= maximum) {
    setVariableData("Priority", String.valueOf(priority+1));
} else {
    setVariableData("Flag", String.valueOf(0));
}
```

Code J.5 Java code for Writing BP’s Start and End Times into a File

```java
import java.io.BufferedWriter;
import java.io.FileWriter;
public class StringTampering {
    public static void test(String start, String end) {
        try {
            FileWriter fstream = new FileWriter("C:\Users\...", true);
            BufferedWriter out = new BufferedWriter(fstream);
            String start1 = start.replace(":", ".");
            String end1 = end.replace(":", ".");
            Float result = Float.valueOf(end1) - Float.valueOf(start1);
            out.write(String.format("%.3f", result));
            out.newLine();
            //close buffer writer
            out.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Code J.6 Java Code for Writing the BP’s Start and End Times into a File

```java
StringTampering.test((String)getVariableData("Start"),(String)getVariableData("End"));
```
Code J.7 Java Code for Executing Database Query Class

```java
String priority1 = (String) getVariableData("Priority");
String group1 = (String) getVariableData("Service-Group");
int priority = Integer.parseInt(priority1);
int group = Integer.parseInt(group1);

String result = DBClass.create(group, priority);
setVariableData("DB-Result", result);

String[] result2 = result.split("\]");
setVariableData("Max-Priority-From-DB", result2[result2.length-1]);
```

Code J.8 Java Code for Executing Dynamic Service Invocation Class

```java
String input = (String) getVariableData("DB-Result");
setVariableData("Invocation-Result", Client2.create(input));
```