LEBANESE AMERICAN UNIVERSITY

A BPEL-BASED FRAMEWORK FOR THE ENFORCEMENT OF WEB SERVICES SECURITY

By

SARA NABIL EL-AYOUBI

A thesis
Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

School of Arts and Sciences
March 2012
Thesis Proposal Form (Annex I)

Name of Student: Sara Ayoubi I.D.#: 201000374

Division: Department of Computer Science and Math.

On (dd/mm/yy) 25/7/2011, has presented a Thesis proposal entitled:

Aspect-BPEL Framework for the Dynamic Enforcement of Web Services Security

in the presence of the Committee members and Thesis Advisor:

Azzam Haddad
(Name and Signature of the Thesis Advisor)

Hadi Othak
(Co-Supervisor)

Rami Al-Hawary
(Name and Signature of Committee Member)

Proposal Approved on (dd/mm/yy): 25/7/2011

Comments/Remarks/Conditions to Proposal:

---

Date: 26 July, 2011 Acknowledged by [Name]
(Dean, School of Arts and Sciences)

cc: Chair
Advisor
Student
File Graduate Studies
Thesis Defense Result Form

Name of Student: Sara Ayoubi
Program / Department: Computer Science & Mathematics
Date of thesis defense: 29th March 2012
Thesis title: A BPEL-BASED FRAMEWORK FOR THE ENFORCEMENT OF WEB SERVICES SECURITY.

Result of Thesis defense:

☐ Thesis was successfully defended. Passing grade is granted
☒ Thesis is approved pending corrections. Passing grade to be granted upon review and approval by thesis Advisor
☐ Thesis is not approved. Grade NP is recorded

Committee Members:

Advisor: Dr. AZZAM MOURAD
(Name and Signature)

Co-Advisor: Dr. HADI OTROK
(Name and Signature)

Committee Member: Dr. RAMZI HARATY
(Name and Signature)

Advisor's report on completion of corrections (if any):

Dr. AZZAM MOURAD

Changes Approved by Thesis Advisor: ________________________________

Date: ________________________

Acknowledged by ____________________________
(Dean, School of Arts and Sciences)

cc: Registrar, Dean, Chair, Thesis Advisor, Student
Thesis Approval Form

Student Name: Sara Ayoubi  I.D. #: 201000374

Thesis Title: A BPEL-BASED FRAMEWORK FOR THE ENFORCEMENT OF WEB SERVICES SECURITY.

Program/Department: Computer Science & Mathematics

School: School of Arts and Sciences

Approved by:
Dr. AZZAM MOURAD

Thesis Advisor:
Dr. HADI OTROK

Co-Advisor:
Dr. RAMZI HARIY

Committee Member

Date 29/03/2012
THESES COPYRIGHT RELEASE FORM

LEBANESE AMERICAN UNIVERSITY NON-EXCLUSIVE DISTRIBUTION LICENSE

By signing and submitting this license, you (the author(s) or copyright owner) grants to Lebanese American University (LAU) the non-exclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video. You agree that LAU may, without changing the content, translate the submission to any medium or format for the purpose of preservation. You also agree that LAU may keep more than one copy of this submission for purposes of security, backup and preservation. You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant LAU the rights required by this license, and that such third-party owned material is clearly identified and acknowledged within the text or content of the submission. IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN LAU, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT. LAU will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Name: Sara Ayoubi

Signature: [Redacted] Date: 29/03/2012
PLAGIARISM POLICY COMPLIANCE STATEMENT

I certify that I have read and understood LAU’s Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me. This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Sara Ayoubi

Signature: [Blacked out] Date: 29/03/2012
ACKNOWLEDGMENTS

This research would not have been possible without the help and assistance of many persons. First I would like to express my gratitude to my supervisor Dr. Azzam Mourad. I am also deeply grateful to the guidance and support of my co-advisor, Dr. Hadi Otrok and the committee member, Dr. Ramzi Haraty. Special thanks go to my mom, my dad, my sisters and brothers and my dear friend Hani Masri.
A BPEL-BASED FRAMEWORK FOR THE
ENFORCEMENT OF WEB SERVICES SECURITY

SARA NABIL EL-AYOUBI

Abstract

In this thesis, we address the problem related to security in a composition of web services, mainly in a BPEL process. This problem emerges due to the monopolization of security at the web service side which causes an enormous overhead when running a process that orchestrates between multiple services. Furthermore, BPEL suffers from a lack of modularity for modeling cross-cutting concerns, thus any changes or modification to the process is a tedious and cumbersome, not to mention the need to deactivate the process throughout the modification phase. Thus, our thesis is dedicated to the introduction of a multilayer framework for the enforcement of security for web services. This approach is based on a synergy between XACML (eXtensible Access Control Markup Language) security policies, Aspect-Oriented Programming (AOP) and composition of web services (BPEL).

This synergy is achieved through the elaboration of a dedicated language called AspectBPEL. The elaborated AspectBPEL language allows specifying security policies as separate components, namely, aspects. These aspects are weaved systematically in the BPEL (Business Process Execution Language) process for the sake of activating the security policies at runtime on specific join points.

In addition, our approach allows specifying the XACML security policies that are used to determine pointcuts in a BPEL process where security is needed. Subsequently, a BPEL flow with the needed security is generated into security AspectBPEL aspects to be weaved in the aforementioned process. The centralization of security at the process level consists on the use of a separate trust authority that adopts an XACML infrastructure.

The main contributions of our approach are: (1) Describing dynamic security policies using a standard language XACML, (2) generating automatically the BPEL aspects of the XACML policies, (3) separating the business and security concerns of composite web services, and hence developing them separately (4) allowing the modification of the dynamic security features and web services composition at run time to integrate, remove and/or update security mechanisms, (5) providing modularity for modeling cross-cutting concerns between web services, (6) centralizing and updating the security measurements at the BPEL side and (7) providing a language and a framework that is fully operational and compatible with any BPEL process regardless of the adopted development environment.

The feasibility and usability of the proposed framework have been verified using two real life case studies: an Online Purchase System (OPS) and a Flight Reservation System (FS). Finally, experimental results and performance analysis are presented to evaluate the proposed framework.

Keywords: Web Services, BPEL, Security, XACML, Aspects Oriented Programming.
Contents

List of Figures xiii

List of Tables xv

Listings xvi

1 Introduction 1

1.1 Motivations and Problem Statement .......................... 1

1.2 Objectives ....................................................... 3

1.3 Approach Overview and Contributions ......................... 4

1.3.1 Aspect-Oriented BPEL Framework for the Dynamic Enforcement of Web Services Security ......................... 5

1.3.2 Extended XACML-AspectBPEL Approach for Composite Web Services Security ........................................ 6

1.4 Thesis Organization ............................................. 7

2 Background & Related Work 9

2.1 Introduction ..................................................... 9
3 AspectBPEL Framework for the Dynamic Enforcement of Web Services Security

3.1 Introduction .................................................. 28
3.2 Illustrative Example: Online Purchase-System Web Services ............ 30
  3.2.1 Online Purchase System Overview ................................ 30
  3.2.2 BPEL Process Architecture ...................................... 31
3.3 Approach Description & Framework Architecture .......................... 32
3.4 AspectBPEL: Aspect-Oriented Language for BPEL ....................... 35
  3.4.1 Grammar ..................................................... 36
  3.4.2 Informal Semantics ............................................ 38
3.5 AspectBPEL Compiler and Framework Implementation .................... 39
  3.5.1 AspectBPEL Compilation ....................................... 40
  3.5.2 AspectBPEL Running and Weaving .............................. 40
3.6 RBAC-OPS: An Authorization Model for a Online Purchase-System Web
  Services ...................................................... 40
  3.6.1 RBAC-OPS Model Definitions .................................. 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7 Case Study: Dynamic Enforcement of the RBAC-OPS Model in the OPS</td>
<td>44</td>
</tr>
<tr>
<td>3.7.1 RBAC-OPS XACML Specification</td>
<td>44</td>
</tr>
<tr>
<td>3.7.2 BPEL Aspects Realizing the RBAC-OPS Model</td>
<td>45</td>
</tr>
<tr>
<td>3.7.3 Experimental Results</td>
<td>46</td>
</tr>
<tr>
<td>3.8 Performance Analysis</td>
<td>47</td>
</tr>
<tr>
<td>3.9 Conclusion</td>
<td>50</td>
</tr>
<tr>
<td>4 Extended XACML-AspectBPEL Approach for Composite Web Services Security</td>
<td>57</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>57</td>
</tr>
<tr>
<td>4.2 Illustrative Example: Flight Reservation System</td>
<td>59</td>
</tr>
<tr>
<td>4.2.1 Flight System Overview</td>
<td>59</td>
</tr>
<tr>
<td>4.2.2 BPEL Process Architecture</td>
<td>60</td>
</tr>
<tr>
<td>4.3 RBAC-FS: An Authorization Model for a Flight-System Web Services</td>
<td>61</td>
</tr>
<tr>
<td>4.4 Approach Description and Architecture</td>
<td>62</td>
</tr>
<tr>
<td>4.5 Approach Design and Implementation</td>
<td>65</td>
</tr>
<tr>
<td>4.5.1 XACML-AspectBPEL Generator</td>
<td>66</td>
</tr>
<tr>
<td>4.5.2 AspectBPEL Compiler/Weaver</td>
<td>67</td>
</tr>
<tr>
<td>4.6 Case Study: Dynamic Enforcement of the RBAC-FS Model in the FS</td>
<td>67</td>
</tr>
<tr>
<td>4.6.1 RBAC-FS XACML Specification</td>
<td>69</td>
</tr>
<tr>
<td>4.6.2 BPEL Aspects Realizing the RBAC-FS</td>
<td>69</td>
</tr>
<tr>
<td>4.6.3 Discussion and Experimental Results</td>
<td>72</td>
</tr>
<tr>
<td>4.7 Conclusion</td>
<td>77</td>
</tr>
</tbody>
</table>
5 Conclusion

5.1 Future Work .................................................. 80

5.2 List of Publications ........................................... 81

Bibliography ......................................................... 83
List of Figures

1 Web Service Process .............................................. 11
2 BPEL Overview .................................................... 13
3 XACML Infrastructure ............................................. 19
4 Aspect Oriented Programming ................................. 21
5 OPS Architecture .................................................. 30
6 OPS BPEL Process ................................................ 32
7 Framework Architecture ........................................... 34
8 Approach Schema Applied on Fig.2 ............................ 35
9 Grammar of AspectBPEL ......................................... 37
10 RBAC-OPS Role Hierarchy ....................................... 43
11 AspectBpel Weaver Snapshot .................................... 51
12 OPS Secure BPEL Process ....................................... 55
13 Performance Analysis ............................................ 56
14 FS Architecture ................................................... 60
15 FS BPEL Process .................................................. 61
16 Approach Schema ................................................ 64
<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Approach Schema</td>
<td>66</td>
</tr>
<tr>
<td>18</td>
<td>XACML-AspectBpel Generator Snapshot</td>
<td>68</td>
</tr>
<tr>
<td>19</td>
<td>FS Secure BPEL Process</td>
<td>74</td>
</tr>
<tr>
<td>20</td>
<td>Performance Analysis - Execution Time</td>
<td>75</td>
</tr>
<tr>
<td>21</td>
<td>Performance Analysis - Compilation/Weaving Time</td>
<td>77</td>
</tr>
</tbody>
</table>
# List of Tables

1. RBAC-OPS Role Hierarchy .............................................. 43
2. Size of the OPSProcess BPEL and WSDL Files .................. 49
3. RBAC-FS Role Hierarchy ............................................... 62
Listings

2.1 Excerpt of a SOAP Request Message .................... 17
2.2 Excerpt of a WSS-SOAP Request Message ............ 18
2.3 A Simple XACML policy - Part I ....................... 22
2.4 A Simple XACML policy - Part II ....................... 23
3.1 Excerpt of XACML-based Access Control Policy for OPS .......................... 52
3.2 Aspect for Authentication ................................. 53
3.3 Excerpt of Generated BPEL Aspect for Access Control ...................... 54
4.1 Excerpt of XACML Policy for FS-Part I ................... 70
4.2 Excerpt of XACML Policy for FS-Part II ................... 71
4.3 Excerpt of Generated Security BPEL Aspects ............... 73
Chapter 1

Introduction

1.1 Motivations and Problem Statement

Security is becoming a hot topic in academia and industry. In this context, customers are expecting security to be delivered out of the box, even on programs that were not designed with security in mind. The challenge is even greater when legacy systems must be adapted to network/web environments, which is the case for web services.

Web services technology has been successful in making business applications available through the internet to a large number of users. It does not just widen the broad of applications accessibility but it is also a catalysis for collaboration between several distributed applications through the composition concept. Nevertheless, the successful deployment of this technology cannot hide the security breaches and threats [6] that web services can be exposed to. Enforcement of web services security is one of the most important duties, that the research community has to perform. This means the design and development of concepts, processes and tools that help in making web services protected from malicious users.
Instances of these security features can be the enforcement of user authentication, access control and data confidentiality in web services.

Conventionally, security measures were embedded statically in the design/code of a web service. This is a problem because, when the security policies and/or verification strategy change, the developer has to go back to the design/code of the web services and update them accordingly. This renders the web services unresponsive during modification. In this context, several standard languages were proposed to enforce web services security policies in a dynamic fashion, among which we find the: Security Assertion Markup Language (SAML) [19], WS-Security [8] and XACML [23].

XACML, SAML, WS-Policy and other standard security languages emerged to offer a standard and dynamic authentication and access control features for web services. In other words, they are concerned with providing outbound message-level security for web services in order to ensure secure communication with external partners.

With the emergence of these standard policy languages, the concern of implementing security at the web service side is alleviated. However, the problem arises at the BPEL level when several distributed and/or independent web services are composed together in a BPEL process to form a complex system. BPEL provides orchestration between several web services that are referred to as "partners". In the current form of BPEL use, BPEL is only given the responsibility of business level orchestration, while message-level security is left to each individual web service to deal with when invoked by the process. Consequently, when a client invokes a BPEL process, which invokes on his behalf every web service in the process. Thus, the security measures (e.g. authentication, access control, credentials verification, etc.) of the same user will be executed at each web service. This will affect
enormously the performance of the BPEL process due to the overhead of running the security verification at each invoke. Moreover, placing security at the process level requires knowledge of the web services’s security requirements which is practically unavailable.

In parallel, with the use of the current BPEL, there is a lack of modularity for modeling cross-cutting concerns and inadequate support for changing the composition at run time. Any change in the environment, addition or removal of partner web services requires static and dynamic adaptation. In other words, if a BPEL runtime process change is required, we have to stop the running process, change the needed web service(s), modify the composition, and then restart. Such a mechanism is cumbersome, error-prone and tedious.

1.2 Objectives

This thesis’s main contribution is the elaboration of a framework for the systematic enforcement of web services security. Our framework offers an elegant and easy to use language, in addition to a compiler to ensure the correctness of the syntax and a weaver for the modular integration of security. More specifically, the objective of our approach are to:

- Address the problem of composite web services security and develop a methodology for the systematic integration of security at the process level.

- Develop a dedicated language for the integration of web services security in a modular fashion.

- Allow the modification of web services compositions at run time to integrate, remove and/or update security mechanisms.
• Implement a framework that allows the compilation and weaving of security aspects in composite web services.

1.3 Approach Overview and Contributions

We propose in this thesis a new framework for the dynamic enforcement of web services security. The proposed framework spans over a new aspect oriented language called AspectBPEL.

In addition, we describe in details the AspectBPEL compiler and its development environment. The AspectBPEL language, allows to specify the security concerns in separate components called aspects. These aspects can then be weaved using the AspectBPEL compiler into the specified BPEL process where the security features are activated at runtime on selected join points. Such approach reduces the security verification at the web services side and provide less security checks at the process level. Furthermore our proposed framework is not only limited to weaving security aspects, it can also be used to weave other functional and non-functional requirements. In the sequel, we present the additional contributions of our new framework.

Moreover, we elaborated an extended approach for the systematic integration of security at the process level using a standard policy language. Our methodology consists of adopting a trust enforcer at the process side that incorporates an XACML infrastructure for authentication and authorization.

In the following, we provide details about the aforementioned contributions.
1.3.1 Aspect-Oriented BPEL Framework for the Dynamic Enforcement of Web Services Security

The main contribution of our work is designing a language for the dynamic integration of web services security. This language is based on aspect oriented programming to allow modular weaving of security aspects at the process level. Thus, it allows the separation between business and security concerns and thus, developing them separately. Our framework offers modularity for modeling cross-cutting concerns, such as security, logging and performance monitoring. The following are the related achievements:

1. Elaborate an approach to separate the business and security concerns of composite web services, and hence developing them separately.

2. Elaborate a language and a framework that is fully operational and compatible with any BPEL process regardless of the adopted development environments (e.g. Eclipse, NetBeans, Oracle).

3. Develop a compiler to ensure the correctness of the syntax and a weaver for the modular integration of cross-cutting concerning.

4. Build a case study to prove the efficiency and effectiveness of our framework.
1.3.2 Extended XACML-AspectBPEL Approach for Composite Web Services Security

The main contribution of this work is the elaboration of an extended approach for BPEL security. Our approach is based on a synergy between XACML, Aspect-Oriented Programming (AOP) and composition of web services. XACML offers the capability of describing the security policies required for a system, while AOP allows to specify the security concerns in separate components called aspects. Our approach is based on the use of a separate trust enforcer, where each partner web service deploys its security policies. At the BPEL side, AspectBPEL aspects will be integrated at selective joint points in order to provide process level security to replace activity-based security caused by the monopolization of security at the web service side.

The main added value to the proposed framework, in addition to the aforementioned achievements presented in section 1.3.1 are:

1. Avoid activity-based security and replace it with a process-level security.

2. Identify selective joint-points for the integration of security aspects in order to avoid unnecessary calls to the XACML components.

3. Describe the security policies using a standard policy language (XACML).

4. Generate BPEL aspects conformed with XACML policies.
1.4 Thesis Organization

The remaining of the thesis is organized as following:

In Chapter 2 we present related works done in the area of web services security and aspect oriented programming. Moreover, we give an introduction on the concepts of web services technology, web services orchestration and Aspect Oriented Programming. We also provide an overview on information security, various security standards and methodologies developed for web services, particularly WS-Security and XACML.

In Chapter 3 we give a thorough description of the elaborated AspectBPEL language, moreover, we discuss in details the AspectBPEL framework in addition to its compiler and weaving engine. To better illustrate the usage of the AspectBPEL grammar, we give an illustrative example with a case study on a "System for Online Purchase". Finally, we conclude the chapter with the results of the performance analysis that proves the appropriateness of our AspectBPEL language for the integration of security aspects at the process level. In addition, we present an analysis of the runtime of our compiler and weaving engine.

In Chapter 4, we present an extension to our AspectBPEL framework that supports the generation of security aspects from a standard policy language: XACML. The XACML-AspectBPEL framework offers the ability to generate security aspects from XACML policies. It also adopts a methodology for the systematic selection and weaving of security behavior at selective join points. We also give an illustrative example of a Flight Reservation System and conclude the chapter with the results of the performance analysis to back-up our methodology and demonstrate the effectiveness of our approach.
In Chapter 5, we summarize briefly the achievements and contributions of this thesis, provide concluding remarks, state the plans for future work, and present the list of publications derived from this thesis.
Chapter 2

Background & Related Work

2.1 Introduction

In this chapter, we provide an overview of all the concepts that compose the components of our framework. Our multi-layer framework is based on a synergy between AOP, XACML, web services and composite web services. First, we will provide an introduction to web services. We will use an example of a Car Manufacturer to better illustrate the use of web services and the advantages of this paradigm. Next, we will explain the concept of BPEL, a business process execution language that allows the orchestration of web services for the composition of complex processes. Moreover, since our thesis is mainly concerned with the topic of security, we will provide a definition of the main security requirements in the area of computer and information security, then provide an introduction of the current security standards applied to web services. The essence of our framework is ensured through the use of Aspect Oriented Programming, thus we will end our background work, with introduction about AOP. Finally, we will conclude this chapter with the related work in the area of web
services security.

2.2 Web Services

Web Services are the latest evolution in distributed computing. This technology allows access to services over network through a combination of Extensible Markup Language (XML), Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description, Discovery and Integration (UDDI). While it may seem very similar to client/server applications, web services don’t offer a GUI to the client and its functionality focuses on sending and processing data in a peer-to-peer fashion. They have also gained a lot of popularity for being both platform and language independent. So what makes web services so popular? To better illustrate the glam of web services, consider the following example: Let’s say you are Ford, the big company that sells cars. Ford has its own website that the customers can access. On the other side, there are a lot of car dealers that also sell Ford cars through their own websites such as caranddriver.com. These car dealers definitely present a huge monetary gain to Ford, and they help raise its sales. However, these brokers use their own database to get the prices of Ford cars. Thus, in case Ford decreases the Price of its Ford Focus, these car dealers won’t know about the price drop which would affect the business of Ford. To help the accuracy of these websites, Ford can grant access to its database by exposing an API that the brokers can invoke and get the exact prices on the fly. Thus, web services consist of exposing APIs that clients can access over a network. These clients need only to know the location of the web service which can be fetched from the UDDI.
The UDDI is the discovery service that keeps a list of available services and their port addresses. Upon receiving the port address from the UDDI, the user needs to know how to invoke the web service. For this purpose, every web service comes with a WSDL. WSDL files are similar to a manual that comes with a system/machine that tells you what kind of features the system/machine offers and how to operate them. Similarly, a WSDL file shows the list of operations that the web service offers, the parameters that each operation needs and the data structure it returns. This XML-Based language is composed of 7 elements:

- Types : Gives the Type of exchanged messages.

- Messages : Defines the request and response messages for each operation, in addition to the elements that compose each message. These elements can be strings, integers,
boolean, date & time, float, double and hexbinary.

- **PortTypes**: Advertises the list of functions that the web service offers with their respective inputs and outputs.

- **Binding**: Specifies a link between <portType> and a protocol (SOAP, HTTP...).

- **Service**: Indicates the port address of each link

- **Port**: Indicates the endpoint of the web service.

- **Operation**: Gives a description of the operation available in the port.

SOAP is a message level protocol for exchanging structured information with web services. It is an XML-Based language composed of a SOAP envelope that encapsulates a SOAP header and a SOAP body. SOAP gained a lot of popularity because it offers extensibility, like WS-Security. Neutrality because it is supported by various transport protocol such as: SMTP, HTTP and TCP. It is also programming language independent.

### 2.3 WS-BPEL

WS-BPEL is an oasis standard executable language for the creation of business processes. These processes are based on the orchestration between several web services to achieve a business goal. For instance, you can create a BPEL process that first invokes a web service to get the weather based on a given zip code, and then invokes another web service that suggests activities to be done based on the weather (shopping if it’s rainy or a trip to the beach if it’s sunny).
BPEL (Business Process Execution Language) 2 is the language that allows the specification of executable business processes. It is xml-based and offers several activities for the manipulation of data sent and received by the interacting partners. It uses WSDL files to interact with each web service, and publishes its own WSDL file to describe its incoming and outgoing messages.

BPEL supports process flow constructs for conditional branching, parallel processes, nested sub-processes, process joins, etc. These constructs [5] are divided into three categories: Actions, Control and Fault.

Among the Actions activities, we have:

- **Empty**: Used for synchronization.
- **Invoke**: Used to invoke a web service.
- **Receive**: Used to specify the partner that sends the request message.
- **Reply**: Used to specify the partner that receives the response message.
- **Assign**: Used to update the value of a variable with new data. The assign activity contains also sub-elements such as Copy, From and To.

Among the Control activities, we have:
• **If**: corresponds to the beginning of a conditional branch in the BPEL process. The If activity contains subelements such as **If, ElseIf, Else**.

• **Pick**: Used to wait for one of several possible messages to arrive or for a time-out to occur. It has the following subelements **onmessage and onalarm**.

• **While**: Used to specify that a child activity is repeated until a certain condition becomes true.

• **Foreach**: Used to iterate its child scope activity exactly N+1 times, where N is equal to the finalCounterValue minus the startCounterValue.

• **RepeatUntil**: Used to define that the child activity is to be repeated as long as the specified condition is true.

• **Wait**: Used to wait for a certain period, or until a certain point in time is reached.

• **Sequence**: Used to define a collection of activities to be performed sequentially in lexical order.

• **Scope**: Used to define a nested activity with its own associated **partnerLinks, messageExchanges, variables, correlationSets, faultHandlers, compensationHandler, terminationHandler and eventHandlers**.

• **Flow**: Used to specify one or more activities to be performed concurrently.

Among the **Fault** activities, we have:

• **Exit**: Used to immediately end a business process.
• **Throw**: Used to generate a fault from inside the process.

• **Rethrow**: Used to rethrow the fault that was originally caught by the immediate enclosing fault handler.

For more computational capabilities, BPEL supports XPath to allow writing expressions and queries, particularly in the control activities, for the composition of sophisticated processes.

WS-BPEL leverages the benefits of the use of a single web service to allow the orchestration of many web services. It offers the ability to compose long-running asynchronous processes with fault detection and compensation activities. Even though such orchestration can be done using any development language such as Java, BPEL allows the abstraction in the composition and infrastructure. Furthermore, BPEL offers the simplicity in composing business processes without the need of traditional coding and programming. Thus any business experts can use BPEL for process automation without any knowledge or experience in the corresponding programming language.

### 2.4 Information Security

With fast evolution of technology, security of online assets becomes a must. Since 1971, many attacks have emerged raising awareness to the existence of malicious attackers, who are willing to invest time and money to breach company security. This can cause huge financial loss, and may lead to publishing company's intellectual property to the public. As Gasser [20] stated once, securing a computer system has traditionally been a battle of
wits: The penetrator tries to find the holes, while the designer tries to close them. Security engineers are in an endless battle to keep their application safe from malicious intruders. And even though, attackers are becoming dangerous as technological equipment keeps emerging to facilitate their work. Many security mechanisms and techniques still stand strong and successfully outwits them.

Information Security is the protection of any computer-related asset from attacks. These attacks can take the form of various threats, where attackers exploit a system’s vulnerability and perform malicious actions. The following are the main security requirements to defend against threats:

- **Authentication**: Corroborating the identity of an entity or source of information
- **Access Control**: Restricting access to resources for privileged entities
- **Data Confidentiality**: Keeping data secret from everyone except those who are authorized to access it.
- **Data Integrity**: Ensuring that data has not been tempered by unauthorized parties.
- **Non-Repudiation**: Preventing denial of previous commitments or actions.

In this thesis, we are mainly concerned with security applied on distribute computing systems, i.e., web services, and the different standards and mechanisms used to apply security into web services. In the sequel, we will discuss two main standards and present their advantages and limitations.
2.5 Web Services Security

With all of the advantages of web services, one of the main hurdles remain security; Particularly because web services use the SOAP protocol for message exchange, which poses a lot of security concerns when sending confidential data. Several Standards appeared to solve the issue of web services security, among these standards we distinguish the WS-Security and XACML policies. In the sequel, we will discuss the aforementioned security standards.

In the real world, we use our driver license to prove our identity, membership cards to prove our right to do certain actions. WS-Security tries to apply the same concept into the context of web services. It is the platform that ensures the exchange of security tokens to authenticate and authorize users in the process of data exchange. It uses the SOAP header to transmit security-related data. For instance, in case the message was signed to ensure integrity, the SOAP header will carry the method used for signing the message and the resultant signature value. Thus, WS-Security is not a security method by itself. Instead it integrates in the SOAP header information about the applied security standards. WS-Security supports various standards for authentication, signatures and encryption such as Kerberos and X.509.

To better illustrate the benefits of using WS-Security, consider the following scenario:

Listing 2.1: Excerpt of a SOAP Request Message

```xml
<wsse:UsernameToken>
  <wsse:Username>scott</wsse:Username>
  <wsse:Password Type="wsse:PasswordText">password</wsse:Password>
</wsse:UsernameToken>
```

Listing 2.1, shows a SOAP message with Scott’s username and password being sent
as plain text. It is rather clear that this message is easy to break. Using WS-Security, we can apply authentication and integrate in the SOAP header security tokens to ensure the integrity of the encrypted soap body.

Listing 2.2: Excerpt of a WSS-SOAP Request Message

<table>
<thead>
<tr>
<th>Listing 2.2: Excerpt of a WSS-SOAP Request Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;wsse:UsernameToken&gt;</code></td>
</tr>
<tr>
<td><code>    &lt;wsse:Username&gt;scott&lt;/wsse:Username&gt;</code></td>
</tr>
<tr>
<td><code>    &lt;wsse:Password Type=&quot;wsse:PasswordDigest&quot;&gt;</code></td>
</tr>
<tr>
<td><code>       KE6QugOpkPyT3Eo0SEgT30W4Keg&lt;/wsse:Password&gt;</code></td>
</tr>
<tr>
<td><code>    &lt;wsse:Nonce&gt;5uW4ABku/m6/S5rnE+L7vg==&lt;/wsse:Nonce&gt;</code></td>
</tr>
<tr>
<td><code>        2002-08-19T00:44:02Z&lt;/wsu:Created&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/wsse:UsernameToken&gt;</code></td>
</tr>
</tbody>
</table>

Listing 2.2 shows Scott’s password digest obtained after applying SHA-1, concatenated with a nonce and a timestamp to ensure message integrity and avoid session replay.

XACML is an oasis standard to incorporate authentication and access control in web services. XACML consists of a policy language and an infrastructure that help define access rights in a web service. The policy language defines who is allowed to access which resource and perform which action on this resource. One XACML policy may contain several policy sets, each defining a new rule for access rights. These rules can have either a permit or a deny effect.

Every rule in a policy has a target, that determines the subject, resource and action to which the rule is applicable. Furthermore, rules can also contain conditions to indicate the constraints under which this rule is applicable.

Listing 2.3 2.4 shows an example of a simple XACML policy. This policy indicates that the subject with an admin role is allowed to access the sampleDomain and perform a write or echo action on a condition that the admin is only granted access from 9am till 5pm (office hours).
In addition to the policy language, Oasis standard defined an XACML infrastructure that consists of four components that can be distributed over the network as web services themselves. These four components work together to provide full authentication and access control to web services:

- **Policy Administration Point (PAP):** Stores XACML access control policies
- **Policy Information Point (PIP):** Hosts attributes about users and services
- **Policy Decision Point (PDP):** Decides about granting and denying access to a resource
- **Policy Enforcement Point (PEP):** Enforces a PDP’s access decision and grants or deny physical access.

To better illustrate the dynamics of the XACML infrastructure, consider Figure 3. In this figure, a user wants to access one of the services offered by a Flight Reservation System. As we have previously mentioned, this system is composed of four separate web services:
services. When the user wants to access one of them, the access request is directed to this web service’s policy enforcement point. The PEP accesses the policy information point that stores information about the user in order to authenticate him. Subsequently, the PIP returns a SAML token with the corresponding authentication response. Next, the PEP accesses the policy decision point in order to retrieve the user’s access rights. The PDP invokes the policy administration point that stores the XACML policies to return the policy of the given resource and the PIP to retrieve the SAML token. With the SAML token and the XACML policy, the PDP returns to the PEP whether this user is granted or denied access to the requested web service. Finally, the PEP either directs the access request to the web service or returns to the application with a response that the user does not have the right to access the requested resource.

2.6 Aspect-Oriented Programming

Aspect Oriented programming is a paradigm that aims to provide modularity by allowing separation of cross-cutting concerns. There are many common features that are scattered around the code, such as logging or metrics collection. In the case of logging, logging code should be placed in different procedures, methods and classes where there is data to be logged. However, this logging code is not really part of the functionality that the class or object model is primarily concerned with. In AOP, features like logging are called cross-cutting concerns, because they "cut across" multiple classes. AOP allows the definition of aspects for each cross-cutting concern. These aspects allows one to pick "join points" within the program, where specific code, also known as "advice", should run at each of
The foundation of AOP is the principle of "Separation of Concerns", where issues that affect and crosscut the application are addressed separately and encapsulated within aspects. Then, these aspects are composed and merged with the core functionality modules into one single application. This process of merging and composition is called weaving, and the tools that perform such process are called weavers. Example of AOP implementations is the AspectJ that offers aspect oriented extension for Java programs. In the sequel, we will present a formal definition of the four primary concepts used in AOP:

- **Cross-cutting concerns**: Is a common secondary requirement shared by several classes in the object model and does not contribute in the logic of the primary functionality of each of these classes, e.g. security, logging, metrics collection, etc.

- **Advice**: Is the additional code that is should be applied to the existing code for the sake of the fulfillment of a cross-cutting concern.

- **Pointcut**: Is represented by a join point match in the code where the injected advice needs to be executed.
• Aspect: Is the combination of pointcuts and advices for the implementation of a cross-cutting concern. e.g. logging aspects, security aspects, etc.

Listing 2.3: A Simple XACML policy - Part I

```xml
1. <?xml version="1.0" encoding="UTF-8"?>
3. <Description>Sample policy</Description>
4. <Target/>
6. <Description/>
7. <Target/>
8. <Subjects>
9. <Subject>
10. <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
11. <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">admin</AttributeValue>
12. <SubjectAttributeDesignator
13. AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
14. DataType="http://www.w3.org/2001/XMLSchema#string"/>
15. </SubjectMatch>
16. <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
17. <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">role</AttributeValue>
18. <SubjectAttributeDesignator
19. AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
20. DataType="http://www.w3.org/2001/XMLSchema#string"/>
21. </SubjectMatch>
22. </Subject>
23. </Subjects>
24. <Resources>
25. <Resource>
27. <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">sampleDomain</AttributeValue>
28. <ResourceAttributeDesignator
29. AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
30. DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
31. </ResourceMatch>
32. </Resource>
33. <Resource>
34. <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
35. <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">SecureEndpoint</AttributeValue>
36. <ResourceAttributeDesignator
37. AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
38. DataType="http://www.w3.org/2001/XMLSchema#string"/>
39. </ResourceMatch>
40. </Resource>
41. </Resources>
```

22
Listing 2.4: A Simple XACML policy - Part II

```xml
41.  <Actions>
42.    <Action>
43.      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
44.        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
45.      </ActionMatch>
46.    </Action>
47.    <Action>
48.      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
49.        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">echo</AttributeValue>
50.      </ActionMatch>
51.    </Action>
52.  </Actions>
53.  </Target>
54.  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
55.    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal">
56.      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
57.        <EnvironmentAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#time" AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time" />
58.      </Apply>
59.      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
60.    </Apply>
61.    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal">
62.      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
63.        <EnvironmentAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#time" AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time" />
64.      </Apply>
65.      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
66.    </Apply>
67.  </Condition>
68.  </Rule>
69.  </Policy>
```

2.7 Related Work

In this section, we will provide an overview on related work in the area of web services security, and the use of AOP to allow modularity for modeling cross cutting concerns.

Web services security is one of the topics that attracted the attention of the research community. From the definition of standards to the publication of research papers, the goal is to provide policies and mechanisms for enforcing web services security. In this
context, several policy standard languages such as Security Assertion Markup language (SAML) [19], WS-Security [8] and WS-XACML [23] were proposed.

SAML [19] is a specification language that is proposed by OASIS. Based on XML, it defines how to specify security credentials, which are represented as assertions. SAML can be used to manage secure sessions between organizations and can leverage several mechanisms such as basic password authentication, SSL and X.509 certificates, etc. A security token is delivered to the requester after successful authentication. This security token allows granting certain permissions to the requester.

WS-Security [8] is a standard that is proposed by IBM, Microsoft, and Verisign. WS-Security is a means for using XML to encrypt and digitally sign SOAP messages. Another feature of WS-Security is that it allows exchanging security tokens for authentication and authorization of SOAP messages.

The web Service eXtensible Access Control Markup Language (WS-XACML) [23] is proposed by OASIS as XML-based language to specify and exchange access control policies. WS-XACML is designed to define authorization policies for principals that are specified using XML.

Bhatti et al. [28] proposed X-RBAC: an XML-based RBAC policy specification framework for enforcing access control in dynamic XML-based web services. The specification uses representations of users, roles and permissions. The two main components of the proposed framework are: the XML and the RBAC processors. The XML processor is implemented in Java using Java API for XML Processing (JAXP). Some modules have the duty to get the DOM instance of parsed XML documents and forward them to the RBAC Processor. The RBAC module is responsible for administration and enforcement of the
policy according to the supplied policy information.

Damiani et al. [10] proposed a design of a web service architecture for enforcing access control policies. They also provide an example of implementation based on the WS-Policy [2], [3] as access control language. The main components of the proposed architecture are: Policy Administration Point (PAP), Policy Evaluation Point (PEP) and Policy Decision Point (PDP). The PAP module is a policy repository that provides an administrative interface for inserting, updating, and deleting policies. The PEP module realizes the enforcement of the policies returned by the PAP module. The access request is granted if at least one policy is satisfied; the access is denied otherwise. A PDP module is the interface between the service and the enforcer module. It is responsible for taking final access control decisions based on the input from the PEP module.

Bertino et al. [14] proposed a RBAC-WS-BPEL framework for defining authorization policies and constraints for WS-BPEL business processes. WS-BPEL is a language for composing web services. To specify authorization policies, the authors used XACML. They introduce the Business Process Constraint Language (BPCL), which can be used to specify authorization constraints. The users are associated with roles as done in Role Based Access Control (RBAC) models.

All the aforementioned approaches target the security policies implementation, deployment and/or verification at the web services side. Moreover, they do not address any of the aforementioned problems at the BPEL level, e.g. dynamic adaptation, services interruption and performance. On the other hand, our approach relies on enforcing the security policies into the BPEL process of the composed web services. These policies are specified as separate components, i.e., AspectBPEL aspects, then integrated systematically using
the AspectBPEL weaver into the BPEL process. The security features can be activated at runtime on selected locations in the BPEL process. This allows to easily update the security measures dynamically when needed, without affecting the business logic of the BPEL process.

In the same line of research, Charfi et al. [11] introduced the AO4BPEL, an aspect oriented extension for BPEL that offers modularity and adaptability to workflow processes. The join points are represented by activities in the BPEL process. Pointcuts are represented in the XPATH language and advices are the BPEL activities to be added. This work has been extended in the Cooperative Aspect Oriented Programming for Executable Business Processes (Co-AOP) tool, which aims at making the aspects reusable [15]. An aspect code is developed for a specific BPEL process, which makes it difficult to reuse. The Co-AOP alleviates that challenge by introducing what is called the Explicit Join Points (EJP). These EJP allow the base code to be aware of the aspect interfaces, and hence improve aspect reusability by decoupling base code and aspects. These aspects are initiated in the base code and described in their advices code, which forces the communication to be parameterized between both the base and aspects code.

The AO4BPEL offers the BPEL process the ability to adapt to future changes in the BPEL process. However, AO4BPEL has few limitations. First, it requires the use of a special orchestration engine to manage the BPEL process, which makes it incompatible with the major adopted BPEL development environments such as Eclipse, NetBeans, etc. Second, their approach imposes overhead since it performs a check on each activity in the process to determine whether or not their aspect code is associated with it. On the other hand, our approach proposes a framework that is fully operational on any BPEL process.
regardless of the adopted development environment. Moreover, our approach reduces enormously the overhead since it is based on intercepting only selective join points, i.e. only those that are associated with the aspect code.

Regarding the usability of AOP for security, the following is a brief overview of the available contributions. In the contexts of programming languages, CSAW was introduced as an AOP language proposed by Cigital labs [1], which is a superset of the C programming language to integrate security in C programs. De Win, in his Ph.D. thesis [12], also discussed an AOP approach that adopts the concept of AOSD for defining security aspects to be weaved within applications. In [9], Ron Bodkin presented a survey on the various security requirements within enterprise applications, with a focus on security cross cutting concerns particularly in authentication and authorization. Furthermore, we find the the Java Security Aspect Library (JSAL), in which Huang et al. [21] introduced a reusable library of security functions implemented in AspectJ. Huang et al.’s work is thus added to the list of contributions in the area of adopting AOP to implement security features.

In [29], Shlowikowski and Ziekinski discussed some security solutions implemented in AspectJ that combines between J2EE, JBoss application server, Java Authentication and Authorization service API (JAAS) and Resource Access Decision Facility (RAD). These approaches are useful to explore the feasibility of using AOP in software security. Hence, we can benefit from their achievements in building our security model.
Chapter 3

AspectBPEL Framework for the Dynamic Enforcement of Web Services Security

3.1 Introduction

This chapter is dedicated to the description of a new framework for the dynamic enforcement of web services security, which is based on a synergy between Aspect-Oriented Programming (AOP) and a composition of web services. This synergy is achieved through the elaboration of a dedicated language called AspectBPEL. The elaborated AspectBPEL language allows to specify security policies as separate components, namely, aspects. These aspects are weaved systematically in the BPEL (Business Process Execution Language) process for the sake of activating the security policies at runtime on specific join points.

Our framework’s main contribution are:
1. Elaborate an approach to separate the business and security concerns of composite web services, and hence developing them separately.

2. Elaborate a language and a framework that is fully operational and compatible with any BPEL process regardless of the adopted development environments (e.g., Eclipse, NetBeans, Oracle).

3. Develop a compiler to ensure the correctness of the syntax and a weaver for the modular integration of cross-cutting concerning.

4. Build a case study to prove the efficiency and effectiveness of our framework.

We verified the usability and feasibility of our approach using an Online Purchase System (OPS) case study. The OPS is composed of several distributed Web services and implements a Role Based Access Control model that indicates the different roles in the system as well as each role access rights. Security policies, corresponding to the aforementioned RBAC model, has been developed and translated into AspectBPEL. Then, by applying the proposed approach and using the AspectBPEL framework and compiler, the aspects are weaved at different join points in the BPEL process of the OPS, providing dynamic activation of the authentication and access control features. Experimental results and performance analysis are presented to defend our propositions.

The rest of this chapter is organized as following: Section 3.2 is devoted to the description of the Online Purchase System (OPS). Section 3.3 is dedicated to the illustration of the proposed approach and framework architecture. Sections 3.4 and 3.5 are devoted to the presentation of the elaborated AspectBPEL language and its corresponding compiler.
In Sections 3.6 and 3.7, we present the RBAC-OPS model and its application using AspectBPEL, in addition to experimental results. In Section 3.8, we discuss the performance analysis performed on our framework. Finally, we conclude the paper in section 3.9.

3.2 Illustrative Example: Online Purchase-System Web Services

In this section, we describe the architecture of the online purchase system Web services and their corresponding BPEL process. The explanation of the illustrative example is needed to understand the proposed approach presented in the following section.

3.2.1 Online Purchase System Overview

Figure 5 explores the interactions between the user, BPEL process and Web services of the online purchase system. As depicted in the figure, the security features are deployed on the Web services side (i.e, not in the BPEL process). This clearly shows that a security check is needed at each Web service(s).

Figure 5: OPS Architecture
Our online purchase system is mainly composed of several distributed Web services, a BPEL process and a graphical user interface that allows the user to make an online purchase, refund purchased items, request for shipment, order new items and apply discount on items. The available services are shown in the system entry page. First, the online purchase service allows the user to buy items online. Second, the refund purchase service lets the user return bought items for a given reason (in case a client is not satisfied with the service). The request for shipment service allows the user to enter the needed information for shipment like location, address, PO.Box, etc... The user has also the ability to order new items and apply discounts to gold member clients.

All users who can access the system have records in the corresponding Web service(s) database. In other words, each user has an ID and a Password stored in the database, in addition to his/her personal information. Each time a user wishes to access one of the online purchase system services, the authentication, access control and policy checker of the selected Web service(s) are invoked to ensure that he/she is not only a valid user, but he/she also has the permission to view the requested information based on the provided policy rules.

3.2.2 BPEL Process Architecture

Figure 6 illustrates a part of the the architecture of the BPEL process of our online purchase system. For space restriction, the figure only explores one service invoke from the BPEL process. This web service is referred to as AnyOPSWebService and it may or may not run with security on the side. The process is invoked when the user requests one of the services.
offered by the Web service. It begins by assigning the client’s input to the OnlinePurchaseService Request message, then it calls the appropriate operation of the requested service and returns the needed info. Finally, the Web service response message is assigned to the BPEL process output variable and forwarded to the client.

![Figure 6: OPS BPEL Process](image)

3.3 Approach Description & Framework Architecture

In this section we describe our approach and the architecture of the AspectBPEL Framework. Aspect Oriented Programming (AOP) [13, 16–18, 27] is one of the most prominent paradigms for integrating non-functional requirements (e.g. security) into software. The main objective of AOP is to have a separation between cross-cutting concerns. This is achieved through the definition of aspects. Each aspect is a separate module in which pointcuts are defined. A pointcut identifies one or more join points. A join point identifies one or many flow points in a program (in our case a program is a BPEL process). At these points, some advices will be executed. An advice contains some code that can alter the process behavior at a certain flow point. The integration of aspects within the
application (BPEL) code is called weaving and is performed through one of the weaving technologies/compilers (e.g., AspectJ [17]).

Security is one of the most important aspects of a software. Generally, developers did not separate between security and business logic code. This means that any change in the security strategy has to be done on the application code, which can have impact on the business logic. AOP helps solve this issue by embedding security in aspects. Aspects allow to precisely and selectively define and integrate security objects, methods and events within application at selected join points, which make them interesting solutions for many security issues. Many contributions [1, 9, 12, 21] [22] [30] [25], have proven the usefulness of AOP for integrating security features into software.

In this context, we propose an aspect-oriented framework for the dynamic enforcement of Web services security. The framework includes a language called AspectBPEL and its corresponding compiler. Our proposition is based on the use of AOP in the BPEL process of the composed Web services. Figure 7 illustrates the framework architecture and depicts all the interactions and flow of controls between its components. The system procedures begin by having security requirements expressed as security policies (e.g. in XACML). The security policies are then specified as BPEL security aspect(s) using the proposed AspectBPEL Language. The formulated aspect(s) together with the selected BPEL process of the composed web services are then passed to the AspectBPEL Weaver, which is responsible of generating the secured BPEL process with all the features specified in the security aspect(s). At the back end of the weaver, we have the AspectBPEL compiler that parses and compiles the BPEL security aspect(s), parses the BPEL process and identifies the selected join points, constructs the data structure required for the static and dynamic weaving
process, and finally weaves the aforementioned aspect(s) into the BPEL process.

The application of the proposed approach on the OPS illustrative example is depicted in Figure 8. It illustrates the BPEL process where the invoke of the security features is embedded in BPEL aspects. By comparing Figure 6 with Figure 8, we can see that the security features are no longer part of the Web services, and hence any modification in the security policies can be reflected in the BPEL aspects that are activated dynamically in the system BPEL process. The BPEL aspects may contain direct security verification to be integrated at some identified join points in the BPEL process, or it may contain an invoke to external Web service(s) that handle the security policies verification. Examples of BPEL security aspects are presented in Section 3.7.

Access control is an instance of security features that requires a run-time check to determine if a user has the right to perform a specific operation. Embedding these features in BPEL aspects allows run-time check and dynamic update in case the verification strategy
changes. Moreover, Web services security is generally represented as a specification of policy rules that are specified in languages such as XACML, WS-Policy, etc. By combining the policies of the Web services into one single process level policy and placing the security policy check at the BPEL side, we can significantly reduce the run-time at each Web service invoke, and thus contribute in reducing the overall execution time of the BPEL process. In fact, through pointcuts and join points, security checks can be injected in certain flow points of the BPEL process. The specification of these pointcuts can be updated at any time without the need to modify the business logic.

### 3.4 AspectBPEL: Aspect-Oriented Language for BPEL

In this section, we present a brief description of the AspectBPEL language. The elaborated language allows the description and specification of security BPEL aspects. It is a language based on advice-pointcut model. We developed AspectBPEL with notations and expressions close to those of the current AOP methodologies but adapted to BPEL. The following are the main features provided by AspectBPEL:
• Automatic BPEL code manipulation such as code addition, substitution, deletion, etc.

• Specification of particular BPEL join points where security code would be injected.

• Description and specification of BPEL aspects.

• Description and specification of reusable BPEL aspects.

• Compatibility with the XPath Language.

3.4.1 Grammar

In this section, we present the syntactic constructs of AspectBPEL and their informal semantics. Figure 9 illustrates the grammar of AspectBPEL.

BPEL Aspect Structure

A BPEL aspect starts with the keyword Aspect, followed by the aspect name. Next comes the aspect code that starts and ends respectively by the keywords BeginAspect and EndAspect.

The aspect code is based on AOP and consists of one or many BPEL_Location_Behavior constructs. Each is composed of a BPEL_Insertion_Point and BPEL_Location_Identifier, where the BPEL behavior code should be injected. A detailed explanation of the components of the aspect code will be illustrated in Section 3.4.2.

The list of AspectBPEL pointcuts, identified in our language as BPEL_Location_Identifier are grouped into three categories: Actions, Controls and Faults that we
BPEL_Aspect ::= Aspect Aspect_Name
BPEL_Aspect_Body
BPEL_Aspect_Body ::= BeginAspect
BPEL_Location_Behavior*
EndAspect
BPEL_Location_Behavior ::= BPEL_Insertion_Point
BPEL_Location_Identifier
BPEL_Behavior_Code
BPEL_Insertion_Point ::= Before
| After
| Replace
BPEL_Location_Identifier ::= Assign <Signature>
| Invoke <Signature>
| Receive <Signature>
| Reply <Signature>
| Empty <Signature>
| If <Signature>
| Pick <Signature>
| While <Signature>
| Foreach <Signature>
| RepeatUntil <Signature>
| Wait <Signature>
| Sequence <Signature>
| Scope <Signature>
| Flow <Signature>
| Exit <Signature>
| Throw <Signature>
| Rethrow <Signature>
BPEL_Condition ::= &&
| Variable <Signature>
BPEL_Behavior_Code ::= BeginBehavior
BPEL_Code_Statements
EndBehavior

Figure 9: Grammar of AspectBPEL
already listed in section 2.3. These pointcuts allow to match all the activities of a BPEL process, where a BPEL code can be inserted before, after or even replace the matched one. The name of each activity is enclosed in < ... > symbols that follows the pointcut. The activity’s name represents the activity’s signature. While some IDEs allow duplicate activity names, it is rather a common practice to adopt unique names for BPEL activities to alleviate ambiguities and keep the process clear.

3.4.2 Informal Semantics

In this Section, we present the informal semantics of the important syntactic constructs in AspectBPEL language.

**BPEL_Location_Behavior**  Is based on the advice-pointcut model of AOP. An aspect may include one or many BPEL_Location_Behavior. Each BPEL_Location_Behavior is composed of the BPEL_Insertion_Point, BPEL_Location_Identifier and BPEL_Behavior_Code.

**BPEL_Insertion_Point**  Specifies the point of code insertion after identifying the location. The BPEL_Insertion_Point can have the following three values: Before, After or Replace. The Replace means remove the code at the identified location and replace it with the new code, while the Before or After means keep the old code at the identified location and insert the new code before or after it respectively.

**BPEL_Location_Identifier**  Identifies the joint point or sets of joint points in the program where the changes specified in the BPEL_Behavior_Code should be applied. The
list of identifiers used in the BPEL_Location_Identifier corresponds to BPEL activities together with their signatures (i.e. name, id and parameters).

**BPEL_Condition**  Identifies the condition that should be validated in order to activate the behavior code. By specifying the name of the variable to be tested and in the signature an operator and a threshold value, we can equip the AspectBPEL with the ability to activate and deactivate a behavior code based on certain conditions.

**BPEL_Behavior_Code**  Contains code written in XPath language, that will be weaved into the BPEL Process. The code will be inserted before/after or replace the location identifier previously stated.

### 3.5  *AspectBPEL* Compiler and Framework Implementation

We used ANTLR V3 Beta 6 and its associated ANTLRWorks V1.4. development environment [26] for the AspectBPEL language description and compiler implementation. The generated Java code allows to parse the BPEL aspect, verify the correctness of its syntax and weaved into the BPEL process. Moreover, we integrated this compiler into a development graphical user interface. The resulting system provides the user with graphical facilities to develop, compile, debug and run BPEL aspects. Figure 11 shows a screenshot of this system where we can see an AspectBPEL compiling, together with the BPEL process to be hardened. The compilation process is divided into two phases that are performed
consequently and automatically. The success of one phase leads to execute the next one. In the sequel, we present and explain these phases.

### 3.5.1 AspectBPEL Compilation

The first phase consists of compiling the AspectBPEL code in order to verify if it respects the aforementioned aspect BPEL structure, this includes automatically parsing and compiling the aspect contents to check the correctness of its syntax, and building the data structure required for the running process. If an error was found, the compilation stops, and an error messages appears on the console indicating the reason that caused the compilation failure.

### 3.5.2 AspectBPEL Running and Weaving

Once the corresponding aspect is compiled successfully, the execution command is constructed based on the information provided in the data structure, which is built during the previous compilation phases. Afterwards, the aspect is weaved with the specified process and the resulted hardened BPEL process is produced.

### 3.6 RBAC-OPS: An Authorization Model for a Online Purchase-System Web Services

In this section, we focus on describing the RBAC-OPS model of our OPS. First, we define the RBAC-OPS model, then we present an excerpt of an XACML-based access control
policy specification for the system. We recall that XACML is an XML based language for policy specification.

### 3.6.1 RBAC-OPS Model Definitions

This model inherits all the components of traditional RBAC models: users, roles, permissions, role hierarchies, user-role assignment, and role-permission assignment relations. Users are assigned to roles and roles are assigned to permissions. An RBAC permission represents the ability to access a certain system service. A user is permitted to execute a service activity if he/she is assigned to a role that has the permission to perform that service. RBAC roles are structured in a hierarchy. More details about describing RBAC model for Web services is presented in [14].

- **$A$:** is the identifier of an activity (e.g., Apply Discount).
- **$R$:** is the set of roles (e.g., Manager, Supervisor, Employee and Staff).
- **$U$:** is the set of potential users.
- **$P$:** is the set of permissions, (e.g., execution of an activity).

**Definition 1: RBAC-OPS Permission**

Let $OPS$ be our System. A $RBAC-OPS$ permission is a tuple $(A_i, Action)$, where $A_i$ is the identifier of an activity in $OPS$ and $Action$ identifies the type of the action that can be performed on activity $A_i$. For example, the tuple $(Apply Discount, execute)$ allows the
authorized user to execute the "Apply Discount" service provided by the Online Purchase System.

**Definition 2: RBAC-OPS Role**

An RBAC-OPS role \( r \) is a set of attribute conditions \( r = \{ ac_i | \text{AttrName}_i \text{ op } \text{AttrValue}_i \} \), where \( \text{AttrName}_i \) identifies a user attribute name, \( \text{op} \) is a comparison or a set of operators, and \( \text{AttrValue}_i \) is a value, a set, or a range of attribute values. Note that the roles \( r \) and \( r' \) might be recognized by the same set of attribute names. However, it is a must that at least one of the values that the attributes of \( r \) and \( r' \) assume must be different. A user can be assigned to only one role while two users identified by the same attributes with the same values are assigned to the same role since we assume that a set of attribute conditions uniquely identifies a role.

**Definition 3: RBAC-OPS Role Hierarchy**

Let \( R \) be a partially ordered set of roles. A role hierarchy defined over \( R \) is the graph of the partial-order relation between the roles in \( R \). If \( r, r_0 \in R \) and \( r < r_0 \), then we say \( r_0 \) dominates \( r \). For instance, our online purchase system consists of four different roles. The highest role is the Manager which has access to all the available services. The supervisor and employee come next in the hierarchy. They have less access rights than the leader but more access rights than the client members. Figure 10 illustrates the role hierarchy of the online purchase management system.
Definition 4: RBAC-OPS User-Role Assignment Relation

Let $U$ be the set of all potential users and $R$ be a partial ordered set of roles. The RBAC-OPS user assignment relation is the set of attributes $UA= \{(u,r) \in U \times R \mid \forall ac_i = AttrName_i op AttrValue_i \in r, \exists attr_j \in CredSet(u)^1 | attr_j = AttrName_i \land ac_i \text{ is evaluated to “true” according to the value of } attr_j \}$. As for the online purchase system, the set of roles are $R=\{Manager, Employee, Supervisor and Client\}$. Assigning users to roles results in a set of attributes that defines the RBAC-Online purchase user assignment relation. For example, in our Online Purchase System, the set of attribute conditions for the Manager role is $r=\{\text{Type= “Manager”, ID= a string of 9 characters, Password= a string of at most 9 characters}\}$; thus a credential set of the user $u=\{\text{Type= “Manager”, ID=“Nadia”, Password= “Moati”}\}$ will be evaluated as “true” and $u$ is assigned to $Manager$. 

---

Figure 10: RBAC-OPS Role Hierarchy

<table>
<thead>
<tr>
<th>Role</th>
<th>Employment Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager</td>
<td>Employment= Manager, ID= a string of at most 9 characters, Password= a string of at most 9 characters</td>
</tr>
<tr>
<td>Employee</td>
<td>Employment= Employee, ID= integer of 9 digits, Password= a string of at most 9 characters</td>
</tr>
<tr>
<td>Supervisor</td>
<td>Employment= Supervisor, ID= integer of 9 digits, Password= a string of at most 9 characters</td>
</tr>
<tr>
<td>Client</td>
<td>Employment= None, ID= integer of 9 digits, Password= a string of at most 9 characters</td>
</tr>
</tbody>
</table>

---

Table 1: RBAC-OPS Role Hierarchy
Definition 5: RBAC-OPS User-Permission Assignment

Let $P$ be the set of permissions of the activity $A_1$ supported by the system, and $RP$ be the set of permission/role assignments. Thus, the RBAC-OPS user-permission assignment relation is the set of attributes $UP= \{(u,p) \in U \times P \mid \exists (u,r) \in UA \mid (r,p) \in RP\}$. For instance, a permission to order new items is assigned to Manager by the $RP$ relation. Thus, a user $u$ can order new items only if he is assigned first to Manager.

3.7 Case Study: Dynamic Enforcement of the RBAC-OPS Model in the OPS

In this section, we present the implementation of the RBAC model for the Online Purchase System that illustrates all the procedures and mechanisms described in our proposed approach for the dynamic enforcement of authentication and access control features in OPS.

3.7.1 RBAC-OPS XACML Specification

Listing 3.1 outlines a summary of the XACML-based access control policy for the OPS. Due to space limitation, we included in the listing the role and permissions of the employee. The others are set in similar way. First, the roles are defined. A general role (root of the hierarchy) is denoted by Manager. It has 2 sub-roles, Employee and Supervisor, and as manager is giving permission to perform any action to any resource. The Employee and Supervisor roles have client as common sub-role and are assigned respectively to PPS:employee:role and PPS:supervisor:role policies. The client role has PPS:client:role
as policy. Each of the permission policies defines the set of permissions related to each role. For instance, the PPS:supervisor:role includes applying a discount.

### 3.7.2 BPEL Aspects Realizing the RBAC-OPS Model

In what follows, we describe BPEL aspects for authentication and access control realizing the aforementioned XACML policy of the RBAC-OPS model.

**User Authentication BPEL Aspect**

For user Authentication, Listing 3.2 illustrates excerpt of an aspect to authenticate the user and assign him role and permission(s). Role and permission descriptions are presented in the aforementioned RBAC-OPS model (see description in section 3.6). We define the location identifier of the Authentication behavioral code to be after the `<bpel:receive...>` construct that receives the request from the user and on condition that the user is accessing the system before 5 pm (closing hours). The Authentication aspect code consists first of assigning the client login info to the Web service request message. Then, the web service is invoked. This latter calls the UserAuthentication operation that loops through the database and returns one of four possible indexes: 1 if the client’s username is invalid, 2 if the client has entered correct username and password, 3 if he entered valid username but incorrect password or 4 if he has no more trials and is not allowed to login anymore. Our system allows the user three trials to enter the correct credentials. After the invoke, an If condition is integrated to check the result returned by the web service. If the user is not authenticated, the web service is invoked again to get the appropriate error string for the returned index. The returned error string is then assigned to the BPEL process output variable and
forwarded to the client. On the other hand, if the user is authenticated, the BPEL process continues its execution.

**BPEL Aspect for Access Control of OPS Services**

Listing 3.3 illustrates an excerpt of the generated aspect that realize the XACML policy of Listing 3.1 to authorize the user access to OPS services. Due to space limitation, we included in the listing the AspectBPEL code that enforce access control on the RefundItems service. The others are set in similar way. As described in the aforementioned RBAC-OPS model (please see description in section 3.6), each user is assigned a role together with its corresponding permission(s). This RBAC model was reflected in the XACML policy and generated into a BPEL aspect code accordingly. The generated aspect code integrates the access control verification before the execution of each invoke activity whose port type and operation were specified in the XACML policy. It begins by an If activity that tests if the value returned by the Authentication partner link response message corresponds to an authorized role. If the check returns a true, then the BPEL process continues by invoking the appropriate web Service. Otherwise it returns an error message and the BPEL process exists.

**3.7.3 Experimental Results**

Weaving the security aspects in Listing 3.2 and Listing 3.3 with the BPEL process of the online purchase system presented in Figure 6 produces the secure BPEL process illustrated in Figure 19. The resulted BPEL process provides dynamic authentication and access control features for the online purchase system. The BPEL process begins by receiving the
client’s login info and the index of the service he requested. After receiving the input, the authentication Web service gets invoked to ensure that the user has valid username and password. If he is not authenticated, the process enters the conditional branch and returns an error string to inform the client of the authentication failure. On the other hand, if the user is authenticated, the process will continue to assign the client’s input to the access control request message. Then, the access control Web service gets invoked to get the client’s permission level and check whether the client has the right to see the requested service. If the access is denied, the process assigns the access control response message to the BPEL output variable and forwards it to the client. Otherwise, the process will proceed to invoke the flight system Web service AnyOPSService and return the client’s requested service.

Verifying the successful integration of the RBAC-OPS security features in the original BPEL code of the online purchase system has been performed through extensive testing. Additional efforts have been spent on verifying that the original functionalities of the system have not been altered. Also, several modifications have been applied to the security policy and reflected dynamically in the corresponding BPEL aspects. Consequently, the modification has been applied dynamically onto the BPEL process, which explores the feasibility and appropriateness of our propositions.

3.8 Performance Analysis

To better demonstrate the effectiveness of our approach, a performance analysis has been performed to measure the variation in the execution time of the BPEL process for three different scenarios: A non-secured BPEL process (NS-BPEL), a BPEL process with security
implemented at the Web Service side (WS-Security) and a BPEL process with AspectBPEL security (AspectBPEL-Security). In the AspectBPEL-Security case, an analysis has been also performed to measure the compilation and weaving time.

The execution time (including the three scenarios) has been measured upon the number of invokes that the BPEL process includes, reflecting the number of participating Web services. The higher the number of invokes are, the bigger and more complex the BPEL process becomes. This helps to demonstrate the scalability of our approach. The execution time has been measured using the Visual Studio Profiling Tool [7]. This tool allows us to read the running time of the process call. Due to the variations in the execution time of a single process call for the same number of invokes, the average of multiple execution has been calculated to reduce the margin of deviation.

Figure 13-a illustrates the following results:

- NS-BPEL process runs at an average of 8493 msec when enclosing 10 web services invokes. It augments linearly to an average of 70000 msec for 100 Web services invokes.

- WS-Security process runs at an average of 9500 msec when enclosing 10 web services invokes. It reaches an average of 90000 msec for 100 Web services invokes.

- AspectBPEL-Security process runs at an average of 9300 msec when enclosing 10 web services invokes. It reaches an average of 72000 msec for 100 Web services invokes. This gives a redeem of more than 10000 msec from the WS-Security process. The achieved results are promising and so close from the NS-BPEL process ones.
Next, we measured the compilation and weaving time of the AspectBPEL-Security framework using the Eclipse Test and Performance Tools Platform [4]. The compilation and weaving runtime represent the time needed to weave the AspectBPEL security aspects into a BPEL process. By looking at Figure 13-B and Figure 13-C, we notice that the compilation and weaving runtime are linear with respect to the size of the AspectBPEL aspect. The size of an aspect is measured by the number of pointcuts/advises.

The aim of this analysis, is not only to show how fast the compilation and weaving functions can be, but also to explore that with the overall overhead, the AspectBPEL-Security enabled process is still faster than a WS-BPEL process. For instance, suppose that the BPEL process with 25 invokes needs an AspectBPEL aspect with 25 different pointcuts, that is, 25 different security behaviors for each one of these invokes. The total runtime of this process will be equal to the \(\text{compilation time} + \text{weaving time} + \text{runtime of the process}\)

\[= 1295.53 + 2165.211 + 15687.758 = 19147 < 19737 \text{ (runtime of WS-Security)}.

Finally, Table 2 shows how the size of the .bpel and .wsdl files of the Online Purchase system process varies in each of the aforementioned scenarios. The size is measured by lines of code. Naturally, the process with AspectBPEL-Security is bigger due to the size of the AspectBPEL code weaved-in in order to implement security on the process side.

<table>
<thead>
<tr>
<th></th>
<th>NS-BPEL</th>
<th>WS-Security</th>
<th>AspectBPEL-Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPSProcess.bpel</td>
<td>178 Lines</td>
<td>264 Lines</td>
<td>272 Lines</td>
</tr>
<tr>
<td>OPSProcess.wsdl</td>
<td>67 Lines</td>
<td>80 Lines</td>
<td>76 Lines</td>
</tr>
</tbody>
</table>

Table 2: Size of the OPSProcess BPEL and WSDL Files
3.9 Conclusion

We presented in this chapter a new approach for the dynamic enforcement of Web services security. Our proposition is based on a synergy between AOP and composition of Web services. It allows the separation between business and security concerns of composite Web services, and hence developing them separately. It also allows the modification of the Web services composition at run time and provides modularity for modeling cross-cutting concerns between Web services. The experiments resulting from developing the RBAC-OPS model and their BPEL aspects, and then deploying them dynamically in the BPEL process of the online purchase system together with the performance analysis demonstrate the feasibility and appropriateness of our propositions. They also illustrate the successful dynamic integration and modification of authentication and access control features in the online purchase system.
Figure 11: AspectBpel Weaver Snapshot
Listing 3.1: Excerpt of XACML-based Access Control Policy for OPS

1. <PolicySet>
2. <!--Role policy set for the manager-->
3. ...
4. <!--Permissions policy set for the manager-->
5. ...
6. <!--Permission to order new items-->
7. ...
8. <!--Include permissions of Employee and Supervisor roles-->
9. <!--Role policy set for the supervisor-->
10. ...
11. <!--Permissions policy set for the supervisor-->
12. ...
13. <!--Permission to apply discount-->
14. ...
15. <!--Include permissions of client role-->
16. ...
17. <!--Role policy set for the employee-->
19. <Target>
20. <Subjects><Subject>
21. <SubjectMatch MatchId="function:anyURI-equal">
22. <AttributeValue DataType="xml:anyURI">employee</AttributeValue>
23. <SubjectAttributeDesignator AttributeId="role" DataType="xml:anyURI"/>
24. </SubjectMatch>
25. </Subject>
26. </Subjects>
27. </Target>
28. <PolicyIdReference>PPS:employee:role</PolicyIdReference>
29. </Policy>
30. <!--Permissions policy set for the employee-->
31. <Policy PolicyId="PPS:employee:role" RuleCombiningAlgId="rule-combine:permit-overrides">
32. <!--Permission to refund items-->
33. <Rule RuleId="Permission:to:refund:items" Effect="Permit">
34. <Target>
35. <Resources><Resource>
36. <ResourceMatch MatchId="function:string-equal">
37. <AttributeValue DataType="xml:string">ProductsWS</AttributeValue>
38. <ResourceAttributeDesignator AttributeId="resource:resource-id" DataType="xml:string"/>
39. </ResourceMatch>
40. </Resource>
41. </Resources>
42. <Actions><Action>
43. <ActionMatch MatchId="function:string-equal">
44. <AttributeValue DataType="xml:string">RefundOrders</AttributeValue>
45. <ActionAttributeDesignator AttributeId="action:action-id" DataType="xml:string"/>
46. </ActionMatch>
47. </Action>
48. </Actions>
49. </Target>
50. <PolicyIdReference>PPS:client:role</PolicyIdReference>
51. </Policy>
52. <!--Role policy set for the client-->
53. ...
54. <!--Permissions policy set for the client-->
55. ...
56. <!--Permission to make purchase-->
57. <!--Permission to request for shipment-->
58. ...
59. </PolicySet>
Listing 3.2: Aspect for Authentication

1. Aspect Authentication
2. BeginAspect
3. 
4. After
5. Receive <receiveInput> && Current_Time < 5:00 pm
6. 
7. BeginBehavior
8. <!--Initialize Authentication Request Message-->
9. <bpel:assign validate="no" name="Assign Input to Authentication message">
10. ...
11. 
12. <!--Invoking the Authentication WS-->
13. <bpel:invoke name="Invoke Authentication" partnerLink="Authentication" operation="userAuthentication" portType="ns:Authentication" inputVariable="AuthenticationRequest" outputVariable="AuthenticationResponse">
14. </bpel:invoke>
15. 
16. <!-- if User is InValid, Reply with error message and Exit Process-->
17. <bpel:if name="If Invalid User" condition="\$AuthenticationResponse.parameters/ns:userAuthenticationReturn!=2">
18. 
19. <bpel:sequence>
20. 
21. <!--Initialize the ErrorString message-->
22. ...
23. <!--Copy the AuthenticationResponse to the ErrorStr -->
24. ...
25. <!--Invoke GetErrorStr WS-->
26. ...
27. <!--Initialize Bpel Output Message-->
28. ...
29. <!--Copy Authentication ErrorStr to Output variable-->
30. ...
31. <!--Return "User Not Authorized" to the Client-->
32. <bpel:reply name="Return ErrorString" partnerLink="client" operation="process" portType="tns:AnyWSProcess" variable="output">
33. </bpel:reply>
34. </bpel:sequence>
35. </bpel:if>
36. EndBehavior
37. EndAspect
1. Aspect AccessControl
2. BeginAspect
3. 
4. Before
5. Invoke <OrderItem>
6. BeginBehavior
7. <!-Accessed by the manager-->
8. ...
9. EndBehavior
10. 
11. Before
12. Invoke <ApplyDiscount>
13. BeginBehavior
14. <!-Accessed by the manager and supervisor-->
15. ...
16. EndBehavior
17. 
18. Before
19. Invoke <RefundItems>
20. BeginBehavior
21. <!-Accessed by the manager and employee-->
22. <bpel:if name="If" odebpelc:lineno="61" xmlns: http="urn: http: namesapce">
24. <bpel:sequence odebpelc:lineno="61"><bpel:assign name="AssignError" odebpelc:lineno="61" validate="no">
25. <bpel:copy odebpelc:lineno="61">
27. <bpel:to odebpelc:lineno="61" part="payload" variable="output"/></bpel:copy>
28. <bpel:copy odebpelc:lineno="61">
29. <bpel:from odebpelc:lineno="61"> "AccessDenied" </bpel:from>
31. <bpel:reply name="ReplyError" odebpelc:lineno="61" operation="process" partnerLink="client" portType="tns: process" variable="output"/></bpel:sequence>
32. </bpel:if>
33. EndBehavior
34. 
35. Before
36. Invoke <BuyItems>
37. BeginBehavior
38. <!-Accessed by the manager, supervisor, employee and client-->
39. ...
40. EndBehavior
41. 
42. Before
43. Invoke <RequestShipment>
44. BeginBehavior
45. <!-Accessed by the manager, supervisor, employee and client-->
46. ...
47. EndBehavior
48. EndAspect
Figure 13: Performance Analysis
Chapter 4

Extended XACML-AspectBPEL

Approach for Composite Web Services

Security

4.1 Introduction

Many literatures have emerged in the domain of web services security and led to the formulation of standard languages for message-level and transport level security for web services. However, few of the work dedicated in this domain tackled the need for process-level security. In this paper, we propose a new approach towards the integration of process-level security for composite web services. It is based on a synergy between XACML (eXtensible Access Control Markup Language) security policies, Aspect-Oriented Programming (AOP) and composition of web services (BPEL). Our approach allows first to specify the XACML security policies, that are used to determine pointcuts in a BPEL process where
security is needed. Subsequently, a BPEL flow with the needed security is generated into security AspectBPEL aspects to be weaved in the aforementioned process. Our approach consists of a dynamic integration of process-level security using a separate trust authority and an integration of security AspectBPEL at selective join points in a process.

The main added value to the proposed framework are:

1. Avoid activity-based security and replace it with a process-level security.

2. Identify selective joint-points for the integration of security aspects in order to avoid unnecessary calls to the XACML components.

3. Describe the security policies using a standard policy language (XACML).

4. Generate BPEL aspects conformed with XACML policies.

To demonstrate the feasibility of our proposition, we have developed a Flight Reservation System (FS) that is composed of several web services. A RBAC (Role Based Access Control) model for the flight reservation system, which we called RBAC-FS, is elaborated and we specified its security requirements using XACML. Afterwards, the web services that implement the security features are developed. Then, the BPEL aspects that integrate the security functionalities into the BPEL process are generated automatically using the elaborated framework and weaved using our compiler introduced in [24]. The devised XACML policies and their corresponding aspects and provide authentication and access control features to the flight reservation system. Case studies and experimental results are presented to defend our propositions.
The rest of the chapter is organized as follows. Section 4.2 is devoted to the description of the flight reservation architecture. In Section 4.3, we present the RBAC-FS model of the flight reservation system. Section 4.4 is dedicated to the illustration of the proposed approach. In Section 4.5 we present our approach’s design and implementation. In Section 4.6, we illustrate the implementation of our propositions in a case study. Finally, we give some concluding remarks in Section 4.7.

### 4.2 Illustrative Example: Flight Reservation System

In this section, we describe the architecture of the flight system Web services and their corresponding BPEL process.

#### 4.2.1 Flight System Overview

Figure 14 explores the interactions between the user, the BPEL process and the Web services of the flight system. As depicted in the figure, the security features are deployed on the Web services side (i.e, not in the BPEL process). This clearly shows that any changes in these security features need a modification in the corresponding Web service.

Our Flight System is mainly composed of three separate Web services, a BPEL process and a graphical user interface that allows the user to request information about the Flight agency staff, get available flights and its monthly revenues, and make a reservation. The system available services are shown in the system main page. First, the financial data service allows the user to request the revenues and expenses of the flight agency for a given month. Second, the Flight Inquiry service returns a list of the available flights including the
Figure 14: FS Architecture

time and date of the departure and arrival plus the name of the airline, and the available seats and tickets price. The Employee Information Service allows the user to view information about the flight system staff by entering their ID number. This information includes the employee’s full name, phone number, email address, post and his office number. Finally, the Make reservation Service enables the user to reserve a seat on a certain flight. All users who can access the system have records in the database. In other words, each user has an ID and a Password stored in the database, in addition to his/her personal information. Each time a user wishes to access one of the flight system services, both the authentication and access control services are invoked to ensure that he/she is not only a valid user, but he/she also has the permission to view the requested information.

4.2.2 BPEL Process Architecture

Figure 15 illustrates a part of the the architecture of the BPEL process of our flight system. For space restriction, the figure only explores one service invoked from the BPEL process. We called this Web service AnyFSWebService. This Web service may or may not run
with security on the side. The process is invoked when the user requests one of the services offered by the Web service. Then, the process assigns the input to the FlightService Request message. Afterwards, the latter calls the appropriate operation of the requested service and returns the needed info. Finally, the Web Service response message is assigned to the BPEL process output variable and then forwarded to the client.

![FS BPEL Process](image)

Figure 15: FS BPEL Process

### 4.3 RBAC-FS: An Authorization Model for a Flight-System Web Services

In this section, we focus on describing the RBAC-FS model of our flight system. The different roles that the FS model adopts are identical to that of the online purchase system presented in section 3.6, thus we will only present the User-Role Assignment Relation depicted in table 3.
Table 3: RBAC-FS Role Hierarchy

<table>
<thead>
<tr>
<th>Role</th>
<th>Employment</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager</td>
<td>Employment= Manager</td>
<td>GetFinancialData, GetEmployeeInfo, GetFinancialInquiries</td>
</tr>
<tr>
<td>Supervisor</td>
<td>Employment= Supervisor</td>
<td>GetEmployeeInfo, GetFinancialInquiries</td>
</tr>
<tr>
<td>Employee</td>
<td>Employment= Employee</td>
<td>GetFinancialInquiries</td>
</tr>
<tr>
<td>Client</td>
<td>Employment= None</td>
<td>MakeReservation</td>
</tr>
</tbody>
</table>

4.4 Approach Description and Architecture

Our proposed approach is based on a synergy between XACML, Aspect-Oriented Programming (AOP) and composition of web services. XACML [23], in addition to other standard languages [8, 19], are very useful for the organized description of complex and composed security policies. They allow to avoid the ad-hoc description of security rules and specify them in XML-based document(s).

The motivation behind the need for a process-level security is to replace the monopolization of web services security by placing it at the BPEL side. This shift will reduce the overhead imposed by restricting security checks at individual invoke activities. Moreover, placing security at the BPEL side requires knowledge of the web services’s security requirements, which is practically unavailable. Thus we have accommodated our approach with a trust authority where each partner web service can securely place their security policies and allow the process to manage security at its side. In addition, we have consolidated our approach with a systematic mechanism of integration of security by identifying selective join points where security needs to be integrated, rather than a dogmatic call for security at each invoke activity.

To better illustrate the importance of process-level security let’s consider our Flight
System illustrated in Figure 14 and let’s consider a process where a leader would want to do a check on the monthly sales then check the flight inquiries to make sure that the sales match with the amount of tickets sold for this month. In this case, the Flight Reservation process will consists of a sequence of two web services invoke: First, the process will invoke the Financial Data web service to retrieve the sales figure of the current month and then it will invoke the flight inquires web service to retrieve the amount of tickets sold. The retrieved information will either be used by the leader to manually ensure that the sales figure matches with the amount of tickets sold, or it will be directed to an automated function to do the required check.

As the process invokes the financial data web service, it will have to go through this service’ policy enforcement point to check whether this user has the right to access the requested data. Once the request for access is identified, the process flow will resume and another access to the policy enforcement point of the flight inquires web service is needed to check whether that same user has access to the requested resource. A closer look to this flow shows the overhead due to the need of security checks at each web service invoke.

Our suggested approach is depicted in Figure 16. The problem consists of moving security from the web service side to the BPEL side in case of composite web services. However, each web service will keep its individual XACML structure to serve requests from other clients that are not directed through the BPEL process. Our approach consists of having each web service deploys its XACML policy at a single Trust enforcer’s policy administration point. The trust enforcer adopts an XACML infrastructure to enforce security at the BPEL level. Alternatively, The web services could provide their security requirements to the BPEL provider to construct these requirements into a single policy to
be deployed at the BPEL’s policy enforcement point. The trust enforcer through its access to the XACML policies will generate AspectBPEL aspects for selective join points in the BPEL process where security is needed. When the BPEL process receives a request, it directs it to the trust enforcer that will launch the chain of calls to the 4 components of the XACML structure in Figure 16. As a result, it either grants access to the service or returns an "access denied" message to the user.

![Figure 16: Approach Schema](image)

Selective join points are the locations identified in the BPEL process where security is required. These locations are found by parsing the XACML policies and matching resources and actions in these policies to invoke activities in the BPEL Process. The advantages of this approach is to avoid security checks on invoke activities where security isn’t required and directing the request immediately to the corresponding service. The need to update the selective join points is only required when the BPEL process’s sequence flow changes and/or the web services’s security requirements change.
To better illustrate the selection of join points consider the XACML policy presented in Listing 4.1 and 4.2.

The permission to get current month sales rule indicates that a leader is allowed access to the FinancialDataWS to get the Current month sales. Since there are specific authentication requirements enforced, this rule will be translated into an AspectBPEL aspects that will be weaved accordingly before the invoke of the Financial Data web service to access the current month sales resources.

On the other hand, the Permission to get the login window does not enforce a need for authentication nor access control and thus this activity is not considered among the selective joint point and will be processed directly.

4.5 Approach Design and Implementation

In the previous chapter, we introduced an approach for dynamic enforcement of composite web services security. The feasibility of our approach was demonstrated by the implementation of a framework that enables the dynamic integration of security aspects in BPEL. We also showed that our framework supports BPEL modularity for separation of crosscutting concerns, in particular between the business and security aspects of BPEL. Moreover, we demonstrated its ability to modify BPEL processes in a dynamic and modular fashion without the need to go back to the BPEL code and manually add or edit activities.

In this chapter, we will present, an extension to our approach that allows the integration of security in BPEL using XACML as a standard policy language. In this context, we will present the structure of our XACML-AspectBPEL platform that is integrated at the Trust
enforcer side for identification of selective join points and generation of the corresponding Security AspectBPEL aspects to be weaved at the BPEL side.

The structure of our XACML-AspectBPEL framework is illustrated in figure 17

![Figure 17: Approach Schema](image)

### 4.5.1 XACML-AspectBPEL Generator

Our XACML parser is developed in Java language based on the DOM parser for xml. It begins by parsing the all the resources identified and actions presented in the policy. Each action on a resource is matched to an invoke activity in the BPEL process side.

For each of the identified location, that were elected to be among the selective join points, an AspectBPEL aspect is generated for each identified location with a call to the Trust Enforcer enclosed within the AspectBehavior tag. The call to the trust enforcer includes information about which action and which resource the user wants to access. If the
user is granted access, the process flow resumes normally, else it returns an error message to the user indicating that he doesn’t have the right to access the requested resource.

4.5.2 AspectBPEL Compiler/Weaver

In order to weave the generated AspectBPEL aspects into the specified BPEL process, we used the AspectBPEL compiler and weaver that we presented in the previous chapter. First, the compiler will parse through the generated aspect to ensure that there are no syntax errors. Next, our AspectBPEL weaver will parse through the join points in order to define the activities where the behavioral BPEL code will be inserted.

Figure 18 shows three snapshots of our XACML-AspectBPEL weaver. Snapshot 1, shows the framework with an opened XACML Policy. In snapshot 2, shows that upon the selection of the "Generate BPEL Aspects" option, the framework will ask to specify the BPEL process to which the aspect will be weaved. Snapshot 3 shows the generated AspectBPEL aspects.

4.6 Case Study: Dynamic Enforcement of the RBAC-FS Model in the FS

In this section, we present the implementation of the RBAC-FS model that illustrates all the procedures and mechanisms described in our proposed approach for the systematic enforcement of security at the process-level.
Figure 18: XACML-AspectBpel Generator Snapshot
4.6.1 RBAC-FS XACML Specification

Listing 4.1 and Listing 4.2 outlines a summary of the XACML-based access control policy for the flight reservation system. Due to space limitation, we included in the listing all the roles and permissions of the FS, but we only elaborated on the GetFlightInquiries from line 71 to line 98 and the GetMonthlySales permissions from line 33 to line 60. The others are set in similar way. First, the roles are defined. A general role (root of the hierarchy) is denoted by Leader and has 2 sub-roles, Manager and Supervisor. The leader is given permission to perform any action to any resource. The Manager and Supervisor roles have Staff as common sub-role and are assigned respectively to PPS:manager:role and PPS:supervisor:role policies. The staff role has PPS:staff:role as policy. Each of the permission policies defines the set of permissions related to each role. For instance, the PPS:supervisor:role includes viewing each staff’s tasks list.

4.6.2 BPEL Aspects Realizing the RBAC-FS

In what follows, we will show the generated XACML-AspectBPEL aspects realizing the aforementioned XACML policy of the RBAC-FS model. Please note that the syntax and constructs of the aspects are specified in AspectBPEL language. Our AspectBPEL framework will compile the aspects. If an error was found, it will return an error message, else it will indicate that the compilation was successful and then weave that latter to the specified BPEL process. Our approach has been tested and we were successfully able to integrate security features into a BPEL process.
Listing 4.1: Excerpt of XACML Policy for FS-Part I

[1]. <PolicySet>
[2]. <!--Permission To Get-Login-Window-->
[3]. <Policy>
[4]. <Rule RuleId="Permission:to:Login" Effect="Permit">
[5]. <Target>
[6].  <Subjects>
[7].   <AnySubject/>
[8].  </Subjects>
[9].  <Resources>
[10]. <Resource>
[12].    <AttributeValue DataType="xml:string">LoginWindow</AttributeValue>
[13].  </ResourceMatch>
[14]. </Resource>
[15]. </Resources>
[16]. </Target>
[17]. </Rule>
[18]. </Policy>
[19]. <!--Defining the role policy set for the Manager-->
[20]. <!--Role policy set for the Manager-->
[21]. <!--Permissions policy set for the Manager-->
[22]. ...
Listing 4.2: Excerpt of XACML Policy for FS-Part II

[59]. </Target>
[60]. </Rule>
[61]. <!--Include permissions of Manager, Supervisor, Employee and Client roles-->
[62]. </Policy>
[63].
[64]. <!--Defining the role policy set for the Employee-->
[65]. </Role policy set for the Employee-->
[66]. ...
[67]. <!--Permissions policy set for the Employee-->
[68]. ...
[69]. <!--Permission to Get-Flight-Inquiries-->
[70]. </Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" PolicyId="RPS:Manager :role" PolicyCombiningAlgId="policy-combine:permit-overrides">
[71]. <Rule RuleId="Permission:To:Get:Flight:Inquiries" Effect="Permit">
[72]. <Target>
[73]. <Subjects>
[74]. <Subject>
[75]. <SubjectMatch MatchId="function:string-equal">
[76]. <AttributeValue DataType="xml:string">Manager</AttributeValue>
[77]. </SubjectMatch>
[78]. </Subject>
[79]. </Subjects>
[80]. </Subjects>
[81]. <Resources>
[82]. <Resource>
[83]. <ResourceMatch MatchId="function:string-equal">
[84]. <AttributeValue DataType="xml:string">FlightInquiries</AttributeValue>
[85]. </ResourceMatch>
[86]. </Resource>
[87]. </Resources>
[88]. </Resources>
[89]. </Actions>
[90]. <Action>
[91]. <ActionMatch MatchId="function:string-equal">
[92]. <AttributeValue DataType="xml:string">GetMonthlyFlightInquiries</AttributeValue>
[93]. </ActionMatch>
[94]. </Action>
[95]. </Actions>
[96]. </Target>
[97]. </Rule>
[98]. <!--Include permissions of Client roles-->
[99]. </Policy>
[100].
[101]. <!--Defining the role policy set for the Supervisor-->
[102]. </Role policy set for the Supervisor-->
[103]. ...
[104]. <!--Permissions policy set for the Supervisor-->
[105]. </Permissions policy set for the Supervisor-->
[106]. ...
[107]. <!--Permission to Get-Employee-Tasks-Lists-->
[108]. ...
[109]. <!--Include permissions of Employee and Client roles-->
[110].
[111]. <!--Defining the role policy set for the Client-->
[112]. </Role policy set for the Client-->
[113]. ...
[114]. </Role policy set for the Client-->
[115]. ...
[116]. </Permissions policy set for the Client-->
[117]. </PolicySet>
Generated BPEL Aspect for Access Control of FS Services

Listing 4.3 illustrates an excerpt of the generated aspect that realize the XACML policy of Listing 4.1 and Listing 4.2. Due to space limitation, we will only show the XACML-AspectBPEL aspects generated for one sequence flow of the entire BPEL Process that we have already described in Section 4.4. This flow allows a leader to get the current monthly sales and the flight inquiries to make sure that the sales match with the amount of tickets sold for the current month. This process flow consists of a sequence of two web services invoke: First, an invoke of the Financial Data web service to retrieve the sales figure of the current month and then an invoke to the flight inquires webservice to retrieve the amount of tickets sold. The retrieved information will be either used by the leader to manually ensure that the sales figure matches with the amount of tickets sold, or it will be directed to an automated function to do the required check.

4.6.3 Discussion and Experimental Results

Generating the security aspects in Listing 4.3 from the XACML policy in Listing 4.1 and 4.2, then weaving them with the BPEL process of the Flight Reservation System presented in Figure 15, produce the secure BPEL process illustrated in Figure 19 (due to the big size of the BPEL process, we show only one service, which we call (AnyFSWebservice). The resulted BPEL process provides dynamic authentication and access control features for the Flight Reservation system. The BPEL process begins by receiving the client’s input and follows the process’s sequence flow. Once the process reaches an invoke activity (that was
Listing 4.3: Excerpt of Generated Security BPEL Aspects

```bpel
[1]. Aspect AccessControl
[2].
[3]. BeginAspect
[4].
[5]. Before
[6]. Invoke <CurrMonthSales>
[7]. BeginBehavior
[8]. <bpel:assign validate="no" name="AccessRequest">
[9]. <bpel:copy>
[10]. <bpel:from><![CDATA["GetCurrentMonthSales"]]> </bpel:from>
[11]. <bpel:to variable="PEPRequest" part="parameters"></bpel:to>
[12]. </bpel:copy>
[13]. <bpel:copy>
[14]. <bpel:from><![CDATA["FinancialDataWS"]]> </bpel:from>
[15]. <bpel:to part="parameters" variable="PEPRequest"></bpel:to>
[16]. </bpel:copy>
[17]. </bpel:assign>
[18].
[19]. <bpel:invoke name="TrustEnforcer" partnerLink="TrustEnforcer" operation="PEP" portType="ns:engine" inputVariable="PEPRequest" outputVariable="PEPResponse">
[20]. </bpel:invoke>
[21]. <bpel:if name="CheckPEPResponse">
[22]. <bpel:condition><![CDATA[$PEPResponse = "Deny"]]> </bpel:condition>
[23]. </bpel:condition>
[24]. </bpel:sequence>
[25]. <bpel:assign validate="no" name="AccessDenied">
[26]. <bpel:from><![CDATA["Access Denied"]]> </bpel:from>
[27]. <bpel:to part="payload" variable="output"></bpel:to>
[28]. </bpel:copy>
[29]. <bpel:reply name="ReplyErrMsg" partnerLink="client" operation="process" portType="tns:process" variable="output">
[30]. </bpel:reply>
[31]. EndBehavior
[32].
[33]. Before
[34]. Invoke <CurrMonthFlightInquiries>
[35]. BeginBehavior
[36]. <bpel:assign validate="no" name="AccessRequest">
[37]. <bpel:copy>
[38]. <bpel:from><![CDATA["GetCurrMonthFlightInquiries"]]> </bpel:from>
[39]. <bpel:to variable="PEPRequest" part="parameters"></bpel:to>
[40]. </bpel:copy>
[41]. <bpel:copy>
[42]. <bpel:from><![CDATA["FlightInquiriesWS"]]> </bpel:from>
[43]. <bpel:to part="parameters" variable="PEPRequest"></bpel:to>
[44]. </bpel:copy>
[45]. </bpel:assign>
[46].
[47]. <bpel:invoke name="TrustEnforcer" partnerLink="TrustEnforcer" operation="PEP" portType="ns:engine" inputVariable="PEPRequest" outputVariable="PEPResponse">
[48]. </bpel:invoke>
[49]. <bpel:if name="CheckPEPResponse">
[50]. <bpel:condition><![CDATA[$PEPResponse = "Deny"]]> </bpel:condition>
[51]. </bpel:condition>
[52]. </bpel:sequence>
[53]. <bpel:assign validate="no" name="AccessDenied">
[54]. <bpel:from><![CDATA["Access Denied"]]> </bpel:from>
[55]. <bpel:to part="payload" variable="output"></bpel:to>
[56]. </bpel:copy>
[57]. </bpel:assign>
[58]. <bpel:reply name="ReplyErrMsg" partnerLink="client" operation="process" portType="tns:process" variable="output">
[59]. </bpel:reply>
[60]. </bpel:sequence>
[61]. </bpel:if>
[62]. EndBehavior
[63]. EndAspect
```
among the selective join points), a call to the trust enforcer is raised in order to determine
the user’s access rights. If the access is denied, the process replies with an "access denied"
message to the client. Otherwise, the process will proceed to invoke the flight reservation
system web service AnyFSWebservice and return to the client the requested resource.

![Diagram of FS Secure BPEL Process]

Figure 19: FS Secure BPEL Process

Verifying the successful integration of the RBAC-FS security features in the original
BPEL code of the flight reservation system has been performed through extensive testing.
Additional efforts have been spent on verifying that the original functionalities of the system have not been altered. Also several modifications have been applied to the security policy and reflected dynamically in the corresponding BPEL aspects. Consequently, the modification has been applied dynamically onto the BPEL process, which demonstrates the feasibility and appropriateness of our propositions.

To better demonstrate the effectiveness of our approach we have also conducted a performance analysis. This analysis allowed us to measure the variation in the execution time between a BPEL process with security on the web service side and a BPEL process with security on the process level.

Figure 20 shows the variation in the process’s runtime. The process execution runtime, has been measured upon the number of invokes that the BPEL process includes, which reflects the number of participating web services. The higher the number of invokes are, the bigger and more complex the BPEL process becomes. This helps demonstrate the scalability of our approach. The execution time has been measured using the Visual Studio Profiling Tool [7]. This tool allows us to read the running time of the process call. Due to
the variations in the execution time of a single process call for the same number of invokes, the average of multiple execution has been calculated to reduce the margin of deviation.

The process execution time chart shows two different lines: a process with security at the web service side and a process with XACML-AspectBPEL security. For the sake of the performance analysis, we built a BPEL process and augmented the number of web services invoke at each run. In the case of security at the web service side, we have developed for each web service an XACML policy and incorporated an XACML infrastructure to ensure proper security as depicted in the policy. However, in the case of security at the BPEL level, we removed the XACML components from the web service side and placed it once at the process side, with all of the web service policies at the PAP component.

Figure20 illustrates the following results:

- A BPEL process with security implemented at the web service level runs at 3225.33 msec for 6 web services invoke, and augments to around 8000 msec for 18 web services invoke.

- A BPEL process with security implemented at the process level runs at 2817.42 msec for 6 web services invoke, and reaches 6521.56 msec when running a process with 18 web services invoke.

These results clearly shows that security at the process level considerably enhances the runtime of the BPEL process.

Furthermore, we measured the compilation and weaving time of the XAML-AspectBPEL framework using the Eclipse Test and Performance Tools Platform [4]. The compilation and weaving runtime represent the time needed to weave the AspectBPEL security aspects
Figure 21: Performance Analysis - Compilation/Weaving Time

into a BPEL process. By looking at Figure 21, we notice that the compilation and weaving runtime are linear with respect to the size of the AspecBPEL aspect. The size of an aspect is measured by the number of pointcuts/advice.

This performance analysis shows that with the XACML-AspectBPEL enhances the overall runtime of the BPEL process, since it alleviate the overhead of sending the security verification at web service side and also reduces unnecessary security checks.

4.7 Conclusion

A novel approach was proposed for the systematic enforcement of security at the process level using a separate trust enforcer with an XACML infrastructure. Our proposition is based on a synergy between XACML, AOP and a composition of web services. It allows the XACML specification of policies and separation between business and security concerns of composite web services, and hence developing them separately. It also allows the modification of the web services composition at runtime and provides modularity for
modeling cross-cutting concerns between web services. The experiments resulting from developing the XACML policies of the RBAC-FS model, generating their corresponding BPEL aspects, and then deploying them in the BPEL process of the flight reservation system, demonstrate the feasibility and appropriateness of our proposition. We also illustrate the successful dynamic integration and modification of access control features in the flight reservation system. Furthermore, we conducted a thorough performance analysis to demonstrate the efficiency of our approach and its scalability for long and complex processes.
Web services is a new paradigm based on distributed computing, and one of the most adopted standards that supports on-demand services. With this new technology comes new concerns. This thesis offers a multi-layer framework that address few concerns related to web services composition. The first part of the thesis was dedicated to the description of our AspectBPEL framework and language. Although Several standards have been employed to implement security at the web service side, these standards does not address the related problems at the process level and poses a lot of overhead in the SOAP messages, and this overhead is particularly exhibited in composite web services. Thus, the second part of our thesis was dedicated to the elaboration of a new approach for security in composite web services.

Thus, the main contributions of our thesis are:

1. Providing a language and a framework that is fully operational and compatible with any BPEL process regardless of the adopted development environments (e.g., Eclipse,
2. Describe the security policies using a standard policy language (XACML) and generate BPEL aspects conformed with these XACML policies.

3. Providing modularity for modeling security cross-cutting concerns between Web services.

4. Reducing as much as possible the security verification at the web services side, and centralizing them at the BPEL side. This provides better performance as explored in our experimental results.

5. Allowing the systematic and selective modification of web services compositions to integrate, remove and/or update security mechanisms.

6. Separating the business and security concerns of composite web services, and hence developing them separately.

5.1 Future Work

Currently, we are working on elaborating an approach for predicting the user’s access rights through the entire process from one path to the XACML structure. As challenging as it is, we see a lot of potentials in this approach since our parser and trust enforcer have access to both the policies and the process. Thus with the right algorithm and methodology, we can equip our trust enforcer with the ability to predict the access rights of each user.

Furthermore, we will adopt the eclipse development platform in order to provide our AspectBPEL framework as a plugin in eclipse. Providing our framework as a plugin will
offer the entire BPEL community with a new approach for developing their business processes and it will allow the separation between business aspect of the process and other non-functional requirements.

5.2 List of Publications

The following is the list of publications derived from the thesis work:

Conference Papers


Submitted Papers

Bibliography


85


