Int. J. Web and Grid Services, Vol. X, No. Y, XXXX

A novel aspect-oriented BPEL framework for the dynamic enforcement of web services security

Azzam Mourad* and Sara Ayoubi

Department of Computer Science and Mathematics,
Lebanese American University, Lebanon
Email: azzam.mourad@lau.edu.lb
Email: sara.ayoubi@lau.edu.lb
*Corresponding author

Hamdi Yahyaoui

Computer Science Department,
Kuwait University, Kuwait
Email: hamdi@sci.kuniv.edu.kw

Hadi Otrok

Department of Electrical and Computer Engineering,
Khalifa University of Science,
Technology and Research, UAE
Email: hadi.otrok@kustar.ac.ae

Abstract: In this paper, we propose a new framework for the dynamic enforcement of composite Web services security, which is based on a synergy between Aspect-Oriented Programming (AOP) and BPEL (Business Process Execution Language). This synergy is achieved through the elaboration of a new language called *AspectBPEL*, which is used to specify security policies as separate components, referred to as aspects, to be weaved systematically in a BPEL process. The injected aspects activate the security policies at runtime on specific join points. Our approach enjoys several additional features such as (1) separating the business and security concerns of composite Web services (2) allowing the update of security mechanisms of composite Web services at run time, (3) providing modularity for modeling cross-cutting concerns between Web services, (4) centralizing some security measurements at the BPEL side and (5) providing a framework fully compatible with any BPEL engine regardless of the adopted development environment.

Keywords: web services; BPEL; security; AOP; RBAC.

Reference to this paper should be made as follows: Mourad, A., Ayoubi, S., Yahyaoui, H. and Otrok, H. (XXXX) 'A novel aspect-oriented BPEL framework for the dynamic enforcement of web services security', *Int. J. Web and Grid Services*, Vol. X, No. Y, pp.xxx–xxx.

Biographical notes: Azzam Mourad is an Assistant Professor in the Department of Computer Science and Mathematics at the Lebanese American University (LAU). He holds PhD degree in Electrical and Computer Engineering from Concordia University, Canada and MSc degree in Computer

Science from Laval University, Canada. The main topics of his current research activities are web services security, web services engineering, aspect-oriented programming, ad-hoc network security, information security, software security and security engineering. He is currently serving as program chair, TPC member and/or reviewers of several international conferences and journals. In the past, he served as Postdoctoral fellow at Concordia University.

Sara Ayoubi is a MSc student in Computer Science at the Lebanese American University (LAU). She holds a bachelor degree in Computer Science from the Lebanese University. The topics of her research activities are security engineering and web services security policies.

Hamdi Yahyaoui is an Assistant Professor in the Computer Science Department at Kuwait University. His research interests include Web services, mobile computing, security, and formal methods. He has a PhD in Computer Science from Laval University, Quebec, Canada.

Hadi Otrok is an Assistant Professor in the Department of Computer Engineering at Khalifa University. He received his PhD in ECE from Concordia University, Montreal, Canada. His research interests are mainly on network and computer security. He has interest on resources management in virtual private networks and wireless networks. His PhD thesis is on 'Intrusion Detection System (IDS)' using Game Theory and Mechanism Design. Before joining Khalifa University, he was holding a postdoctoral position at the École de technologie supérieure. He is serving as a technical program committee member for different international conferences and regular reviewer for several journals.

1 Introduction

Web services technology has been successful in making business applications available through the internet to a large number of users. It did 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 that a Web service can be exposed to. Enforcement of Web services security is one of the most important duties, which requires a special attention from the research and industrial communities through the design and development of concepts, processes and tools that help protect Web services from malicious users and satisfy security policies. Instances of these security measures can be the enforcement of authentication, access control and data confidentiality related to the interactions between users and Web services.

In this context, several standards for Web services security were proposed, among which we find: Security Assertion Markup Language (SAML) (Lockhart, 2008), WS-Security (Atkinson, 2006) and XACML (Moses, 2011). However, few research initiatives explored the area of security for composite Web services systems. Our work, in this paper, is focused on WS-BPEL, which is one example of such systems. 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. 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, it 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 security requirements which is practically unavailable. Furthermore, in the current form of BPEL processes, there is a lack of modularity for modelling 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 there is a need to change the BPEL process, 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.

To address the aforementioned problems and overcome the current BPEL weaknesses, we propose, in this paper, an aspect-oriented framework for the dynamic enforcement of composite Web services security. The proposed framework spans over a new aspect oriented language called *AspectBPEL*. 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 to be activated at runtime on selected join points. Such approach reduces the security verification at the Web services side and provides security 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 order to provide modularity for modelling cross-cutting concerns between Web services (such as logging or performance monitoring). Additional contributions of our new framework are summarised as follows:

1 Providing modularity for modelling security cross-cutting concerns between Web services.

2 Allowing the systematic and selective modification of a BPEL process to integrate, remove and/or update security mechanisms.

3 Reducing as much as possible the security verification at the Web services side, and centralising them at the BPEL side. This provides better performance as shown by our experimental results.

4 Providing a language and a framework that is fully operational and compatible with any BPEL engine regardless of the adopted development environments.

We verified the usability and feasibility of our approach and framework using an Online Purchase System (OPS) case study. OPS is composed of several distributed Web services and it implements a Role Based Access Control model that indicates the different roles in the system as well as each role's access rights. Security policies corresponding to the aforementioned RBAC model has been developed and translated into AspectBPEL. Next, by applying the proposed approach and using the AspectBPEL framework, the aspects are weaved at different join points in the OPS BPEL process. Thus, providing dynamic activation of authentication and access control features. Experimental results and performance analysis are presented to defend our propositions. A demo of AspectsBPEL Framework is provided at the following link: http://youtube/K6XAebSnH2E.

The rest of the paper is organised as following: Section 2 is devoted to the description of OPS. Section 3 is dedicated to the illustration of the proposed approach and framework

architecture. Section 4 is devoted to the presentation of the elaborated AspectBPEL language and its corresponding compiler. In Sections 5 and 6, we present the RBAC-OPS model and its application using AspectBPEL, in addition to experimental results. In Section 7, we discuss the performance analysis performed on our framework. In Section 8, we discuss the related work and finally we provide some concluding remarks in Section 9.

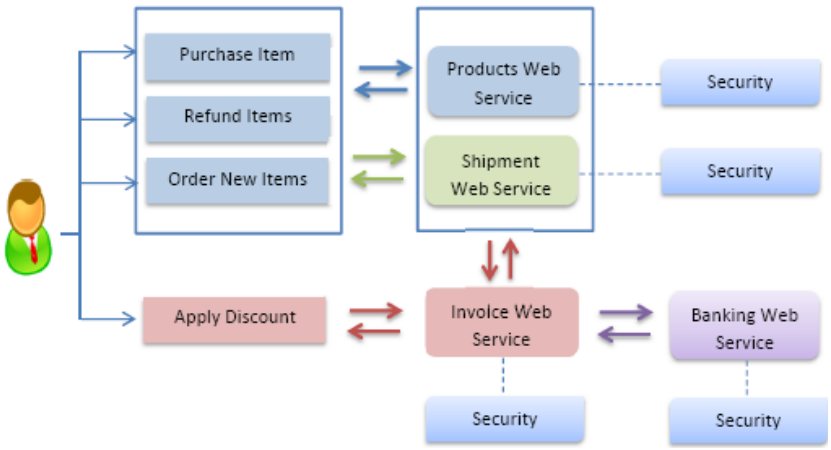
2 Illustrative example: OPS web services

In this section, we describe the architecture of OPS and the corresponding BPEL process. The explanation of the illustrative example is needed to understand the proposed approach presented in the following section.

2.1 OPS overview

Figure 1 explores the interactions between the user, the BPEL process and Web services of OPS. As depicted in the figure, the security features are deployed on 4 the Web services side. This clearly shows that a security check is needed at each Web service invoke.

Figure 1 OPS architecture (see online version for colours)



OPS is mainly composed of several distributed Web services, a BPEL process and a user interface that allows a 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 give discounts to VIP clients.

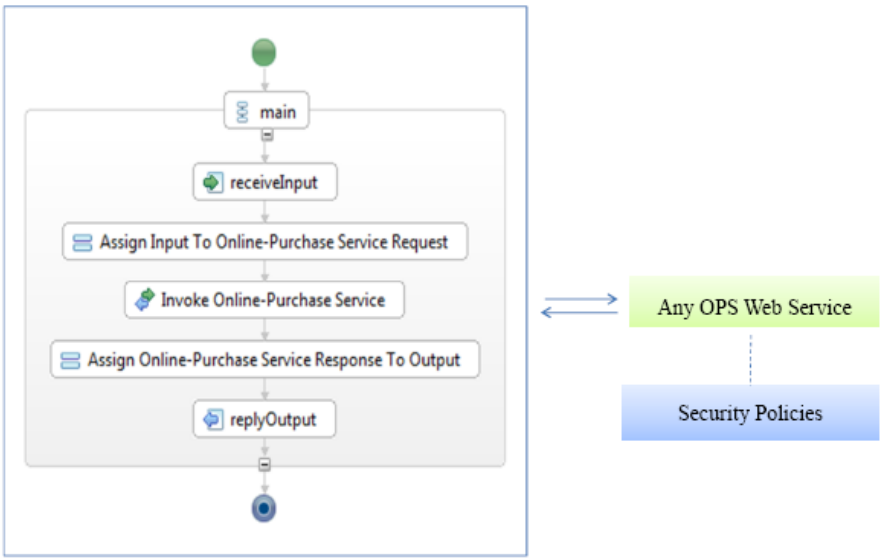
All users that can access the system have records in the corresponding Web service(s) database. In other words, each user has an ID and a Password, in addition to his/her

personal information. Each time a user wishes to access one of OPS 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.

2.2 BPEL process architecture

Figure 2 illustrates part of the BPEL process that represents OPS. For space restriction, the figure only shows one Web service invoke activity in 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 user'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's response message is assigned to the BPEL process output variable and returned to the user.

Figure 2 OPS BPEL process (see online version for colours)



3 Approach description & framework architecture

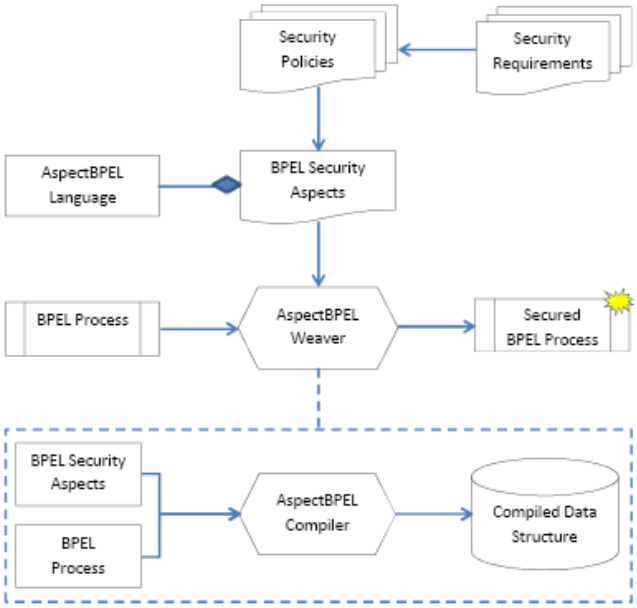
In this section, we describe our approach and the architecture of the AspectBPEL Framework. Aspect Oriented Programming (AOP) (Pavlich-Mariscal et al., 2007; Fuentes and Sanchez, 2006; Evermann, 2007; Kiczales et al., 1997; Kiczales et al., 2001) is one of the most prominent paradigms for integrating 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 behavioural code that can alter the process behaviour 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. Kiczales et al., 2001).

Security is one of the most important aspects of a software. Generally, developers do 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 on solving this issue by embedding security in aspects. Aspects allow to precisely and selectively define and integrate security objects, methods and events within applications at selected join points, which make them interesting solutions for many security issues. Many contributions (Shah, 2003; DeWin, 2004; Bodkin, 2004; Huang et al., 2004; Sun et al., 2009; Wu-Lee and Hwang, 2010) have proven the usefulness of AOP for integrating security features into software.

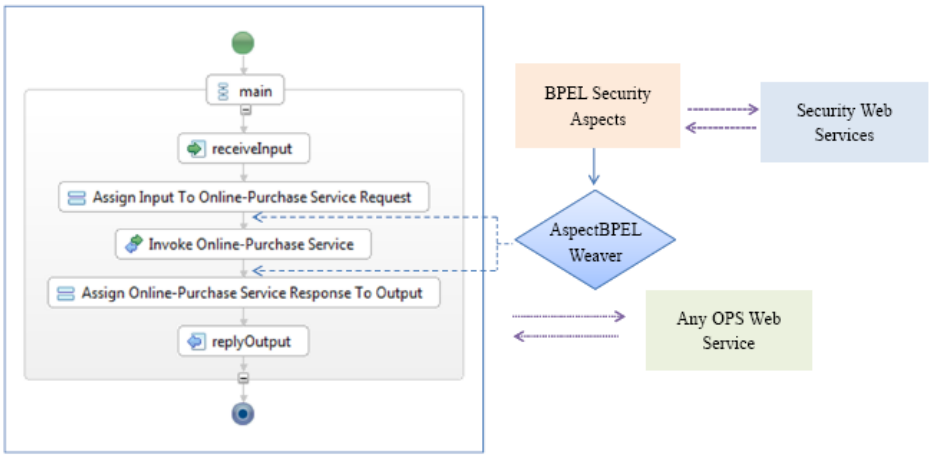
In this context, we propose an aspect-oriented framework for the dynamic enforcement of Web services security. The framework includes a language called AspectBPEL and its corresponding compiler. Our proposition is based on the use of AOP in a BPEL process, which is composed of several Web services. Figure 3 illustrates the framework architecture and depicts all the interactions and control flows between its components. The system procedures begin by having security requirements expressed as security policies (e.g. in XACML). The security policies are then described as BPEL security aspects using the proposed AspectBPEL Language. The formulated aspects 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 aspects. At the back end of the weaver, we have the AspectBPEL compiler that parses and compiles the BPEL security aspects, then parses the BPEL process to locate the insertion points, and finally weaves the aforementioned aspects into the BPEL process.

Figure 3 Framework architecture (see online version for colours)



The application of the proposed approach on the OPS illustrative example is depicted in Figure 4. It illustrates the BPEL process where security features are embedded as BPEL aspects. By comparing Figure 2 with Figure 4, we can see that the security features are no longer at the Web service side, and hence any modification in the security policies will be reflected in aspects that will be activated dynamically in the 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 6.

Figure 4 Approach schema applied on Figure 2 (see online version for colours)



4 AspectBPEL: aspect-oriented language for BPEL

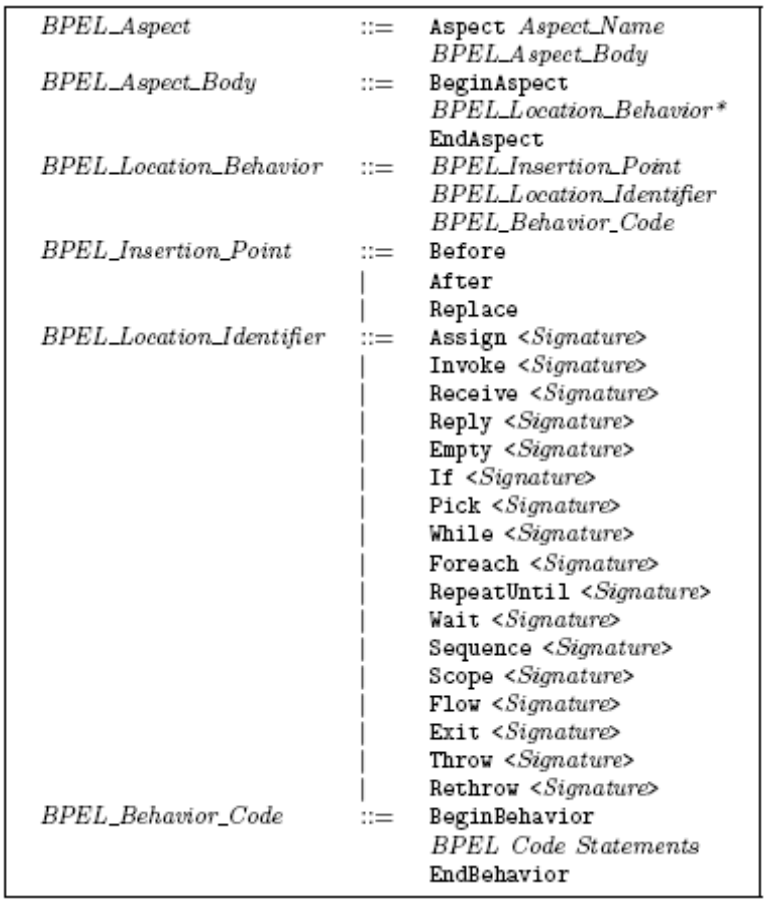
In this section, we present a description of the AspectBPEL language. The elaborated language allows the specification of security BPEL aspects. It is a language based on advice-pointcut model. We developed AspectBPEL with notations and expressions close to those of the current AOP methodologies but 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 reusable BPEL aspects.
- Compatibility with any BPEL engine.

4.1 Grammar

In this section, we present the syntactic constructs of AspectBPEL and their informal semantics. Figure 5 illustrates the grammar of AspectBPEL.

Figure 5 Grammar of AspectBPEL



4.1.1 BPEL aspect structure

An AspectBPEL 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 Behaviour constructs. Each is composed of a BPEL_Insertion_ Point and BPEL_Location_Identifier, where the BPEL behaviour code should be injected. A detailed explanation of the components of the aspect code will be illustrated in Section 4.2. The list of AspectBPEL pointcuts, identified in our language as BPEL_Location_Identifier are grouped into three categories: *Actions, Controls* and *Faults*. 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.

4.2 Informal semantics

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

4.2.1 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.

4.2.2 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 removing the code at the identified location and replacing 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.

4.2.3 BPEL_Location_Identifier

Identifies the join point or sets of join 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). In the sequel, we illustrate the activities used for identifying locations, which are divided into three categories: *Actions, Control* and *Fault*.

Among the *Actions* pointcuts, we have:

Empty <Signature>: Used for synchronisation.

Invoke <Signature>: Used to invoke a Web service.

Receive <Signature>: Used to specify the partner that sends the request message.

Reply <Signature>: Used to specify the partner that receives the response message

Assign <Signature>: Used to update the value of a variable with new data. The assign activity contains also subelements such as **Copy, From and To**.

Among the *Control* pointcuts, we have:

If <Signature>: corresponds to the beginning of a conditional branch in the BPEL process. The *If* activity contains subelements such as **If, ElseIf, Else**.

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

While <Signature>: Used to specify that a child activity is repeated until a certain condition becomes true.

Foreach <Signature>: Used to iterate its child scope activity exactly N+1 times where N is equal to the *finalCounterValue* minus the *startCounterValue* that are initialised within the foreach activity.

RepeatUntil <Signature>: Used to define that the child activity is to be repeated as long as the specified condition in this activity is true.

Wait <Signature>: Used to wait for a certain period, or until a certain point in time is reached.

Sequence <Signature>: Used to define a collection of activities to be performed sequentially in lexical order.

Scope <Signature>: Used to define a nested activity with its own associated *partnerLinks, messageExchanges, variables, correlationSets, faultHandlers, compensation Handler, terminationHandler and eventHandlers*.

Flow <Signature>: Used to specify one or more activities to be performed concurrently.

Among the *Fault* pointcuts, we have:

Exit <Signature>: Used to immediately end a business process.

Throw <Signature>: Used to generate a fault from inside the process.

Rethrow <Signature>: Used to rethrow the fault that was originally caught by the immediate enclosing fault handler.

4.2.4 BPEL_Behavior_Code

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

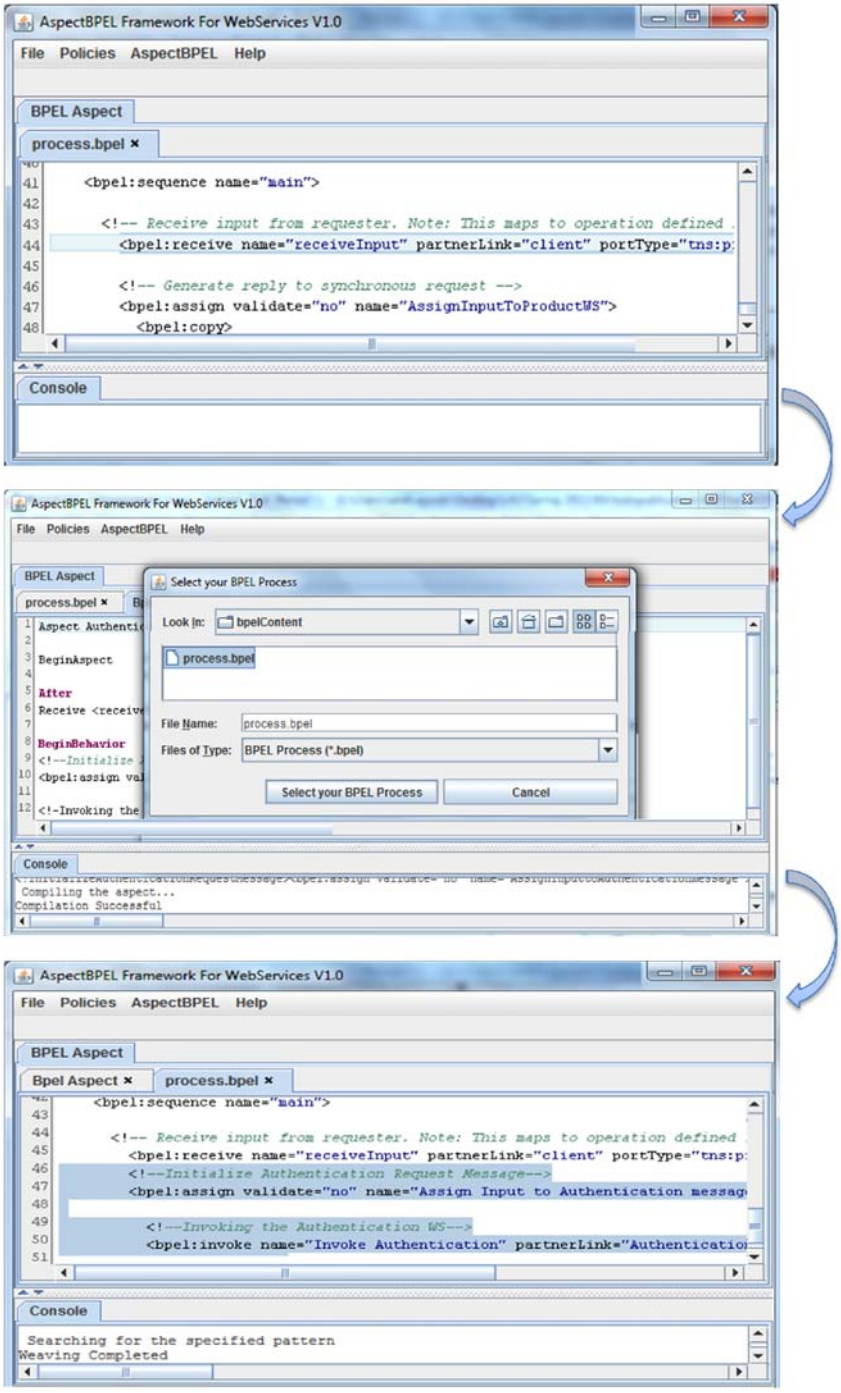
4.3 AspectBPEL compiler and framework implementation

The AspectBPEL grammar and compiling/weaving engine were developed using ANTLR V3 Beta 6 and its associated ANTLRWorks V1.4. development environment (see http://www.antlr.org/). The AspectBPEL engine is equipped with a compiler that verifies the correctness of an AspectBPEL aspect and a weaver to merge the aspect in a BPEL process. Moreover, we integrated this compiler into a graphical user interface. The resulting system provides the user with graphical facilities to develop, compile, debug and run BPEL aspects. Figure 6 shows a screenshot of this system where we can see an AspectBPEL compilation and 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 allows the execution of the other. In the sequel, we present and explain these phases.

4.3.1 AspectBPEL compilation

The first phase begins by compiling the AspectBPEL code in order to verify if it respects the AspectBPEL grammar. This grammar check consists of automatically parsing and compiling the aspect to check the correctness of its syntax, and building a data structure of insertion points, location identifiers and behaviour code to be used in the weaving phase. If an error is found, the compilation stops, and an error message is displayed on the console to indicate the error that caused compilation failure.

Figure 6 AspectBpel weaver snapshot (see online version for colours)



4.3.2 AspectBPEL running and weaving

Once the AspectBPEL aspect is compiled successfully, the built data structure will be used to weave the aspect's BPEL Behaviour Code at the corresponding BPEL Insertion Points. This phase will result in the production of a secured BPEL process.

5 RBAC-OPS: an authorisation model for OPS

In this section, we focus on describing the RBAC-OPS model of 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 a XML based language for policy specification.

5.1 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. A 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 Paci et al. (2008).

The below acronyms will be used for the RBAC model description:

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

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 *Ai*. For example, the tuple (*Apply Discount*, *execute*) allows the authorised user to *execute* the Apply Discount service provided by OPS.

Definition 2: RBAC-OPS Role

A RBAC-OPS role r is a set of attribute conditions $r = \{ac_i \mid ac_i = AttrName_i \ op \ AttrValue_i\}$, where $AttrName_i$ identifies a user attribute name, *op* is a comparison or a set of operators, and $AttrValue_i$ is a value, a set, or a range of attribute values. Note that the roles r and r' might be recognised 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, OPS 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 7 illustrates the role hierarchy of OPS.

Figure 7 RBAC-OPS role hierarchy

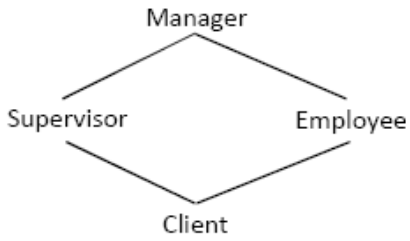


Table 1 RBAC-OPS role hierarchy

R	
Manager characters	{Employment= Manager, ID= a string of at most 9 Password= a string of at most 9 characters}
Employee	{Employment= Employee, ID= integer of 9 digits, Password= a string of at most 9 characters}
Supervisor	{Employment= Supervisor, ID= integer of 9 digits, Password= a string of at most 9 characters}
Client	{Employment= None, ID= integer of 9 digits, Password= a string of at most 9 characters}

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 x R | $\forall$ ac_i = $AttrName_i$ *op* $AttrValue_i \in r$; $\exists$ $attr_j \in$ CredSet(u)[1] | $attr_j$= $AttrName_i \wedge ac_i$ is evaluated to "true" according to the value of $attr_j$}. As for OPS, the set of roles are R={*Manager*, *Employee*, *Supervisor* and *Client*}. Assigning roles to users results in a set of attributes that defines the RBAC-Online purchase user assignment relation. For example, in OPS, the set of attribute conditions for the Manager role is r= {Type= "Manager", ID= a string of 9 characters, Password= a string of at most 9 characters}; thus a credential set of the user u={Type= "Manager", ID="Nadia", Password= "Moati"} will be evaluated as "true" and u is assigned to *Manager*.

Definition 5: RBAC-OPS user-permission assignment

Let *P* be the set of permissions of the activity *A*1, 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) ∈ U x P | ∃ (u,r) ∈ UA | (r,p) ∈ 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*.

6 Case study: dynamic enforcement of the RBAC-OPS model in OPS

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

6.1 RBAC-OPS XACML specification

Figure 8 outlines a summary of the XACML-based access control policy for OPS. Due to space limitation, we included in the listing the role and permissions of the employee. The other permissions are set in a 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". The manager is given permission to perform any action to any resource, while 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.

6.2 BPEL aspects realising the RBAC-OPS model

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

6.2.1 User authentication BPEL aspect

For user Authentication, Figure 9 illustrates excerpt of an aspect used to authenticate the user and assign him a role and permission(s). Role and permission descriptions are presented in the aforementioned RBAC-OPS model (see description in Section 5). We define the location identifier of the Authentication behavioural code to be after the <bpel:receive...> construct that receives the initial request from the user. The Authentication aspect code consists of invoking the "UserAuthentication" operation that returns a token to indicate if the user's credentials are valid. After the invoke, an If condition is integrated to check the result returned by the Web service. If the user is not authenticated, the BPEL process exits and returns an error message to the user. On the other hand, if the user is authenticated, the BPEL process continues its execution.

Figure 8 Excerpt of XACML-based access control policy for OPS

```
<PolicySet>
<!--Role policy set for the manager-->
...
<!--Permissions policy set for the manager-->
...
  <!--Permission to order new items-->
    ...
    <!--Include permissions of Employee and Supervisor roles-->
<!--Role policy set for the supervisor-->
...
<!--Permissions policy set for the supervisor-->
...
    <!--Permission to apply discount-->
    ...
    <!--Include permissions of client role-->
    ...
<!--Role policy set for the employee-->
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" PolicyId="RPS:
     employee:role" PolicyCombiningAlgId="policy-combine:permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="function:anyURI-equal">
          <AttributeValue DataType="xml:anyURI">employee</AttributeValue>
          <SubjectAttributeDesignator AttributeId="role" DataType="xml:anyURI"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <!--Include permissions of employee role-->
  <PolicyIdReference>PPS:employee:role</PolicyIdReference>
</Policy>
<!--Permissions policy set for the employee-->
  <Policy PolicyId="PPS:employee:role" RuleCombiningAlgId="rule-combine:permit-
       overrides">
  <!--Permission to refund items-->
  <Rule RuleId="Permission:to:refund:items" Effect="Permit">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="function:string-equal">
          <AttributeValue DataType="xml:string">ProductsWS</AttributeValue>
          <ResourceAttributeDesignator AttributeId="resource:resource-id"
               DataType="xml:string"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="function:string-equal">
          <AttributeValue DataType="xml:string">RefundItems</AttributeValue>
          <ActionAttributeDesignator AttributeId="action:action-id" DataType="
               xml:string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  </Rule>
  <!--Include permissions of client role-->
  <PolicyIdReference>PPS:client:role</PolicyIdReference>
</Policy>
<!--Role policy set for the client-->
...
<!--Permissions policy set for the client-->
  ...
  <!--Permission to make purchase-->
  ...
  <!--Permission to request for shipment-->
    ...
</PolicySet>
```

Figure 9 Aspect for authentication

```
Aspect Authentication

BeginAspect

Before
After <receiveInput>

BeginBehavior
<!-Initialize Authentication Request Message-->
<bpel:assign validate="no" name="Assign Input to Authentication message">
   ...

<!-Invoking the Authentication WS-->
<bpel:invoke name="Invoke Authentication" partnerLink="Authentication" operation
    ="userAuthentication" portType="ns:Authentication" inputVariable="
    AuthenticationRequest" outputVariable="AuthenticationResponse">
</bpel:invoke>

<!-- if User is Invalid, Reply with error message and Exit Process-->
<bpel:if name="If Invalid User">      <bpel:condition><![CDATA[(
    $AuthenticationResponse.parameters/ns:userAuthenticationReturn!=True)]]>
</bpel:condition>
<bpel:sequence>

<!-Initialize the ErrorString message-->
...
<!-Copy the AuthenticationResponse to the ErrorStr -->
...
<!-Invoke GetErrorStr WS-->
...
<!-Initialize Bpel Output Message-->
...
<!-Copy Authentication ErrorStr to Output variable-->
...
<!-Return "User Not Authenticated" to the Client"-->
<bpel:reply name=" Return ErrorString" partnerLink="client" operation="process"
    portType="tns:AnyWSProcess" variable="output">
</bpel:reply>
</bpel:sequence>
</bpel:if>
EndBehavior
EndAspect
```

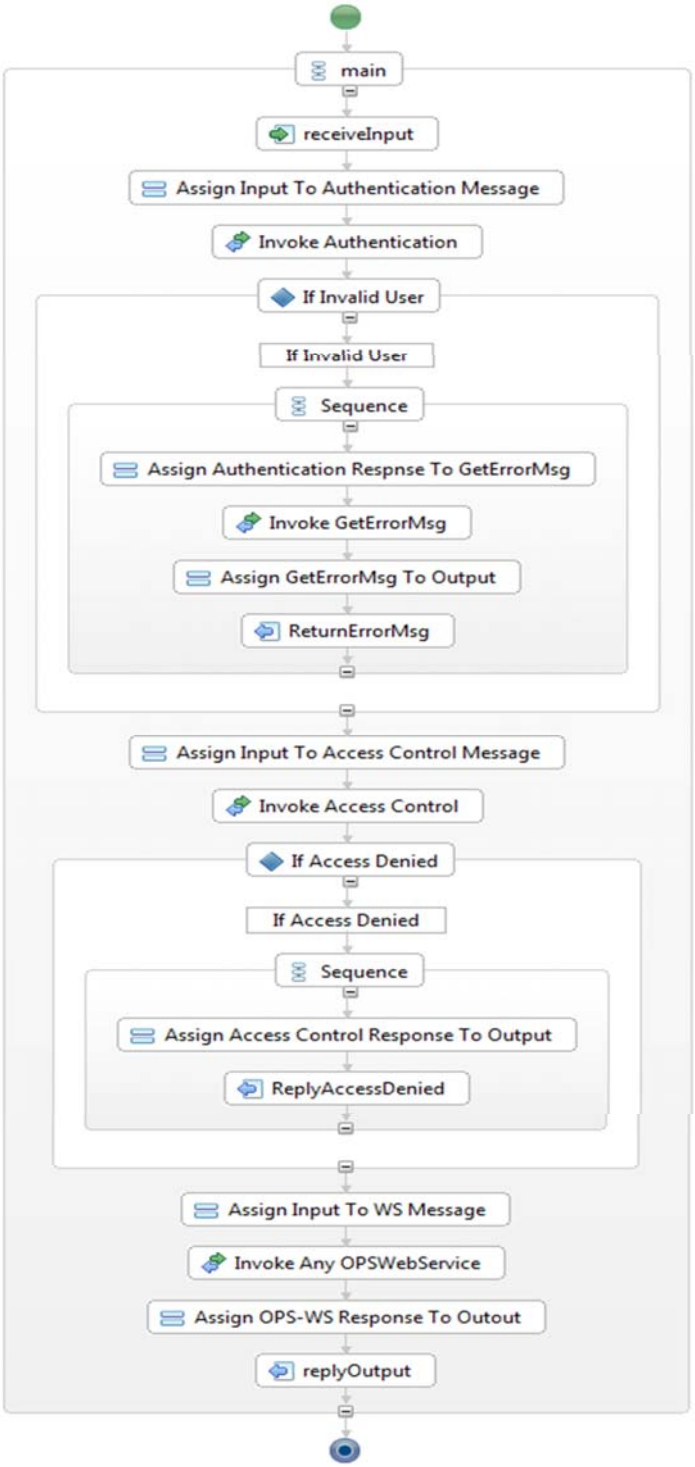
6.2.2 BPEL aspect for access control of OPS services

Figure 10 illustrates an excerpt of the generated aspect that realises the XACML policy of Figure 8 to authorise the user to access OPS services. Due to space limitation, we included in the listing the AspectBPEL code that enforces access control on the RefundItems service. The others are set in a similar way. As described in the aforementioned RBAC-OPS model (please see description in Section 5), 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. Since an invoke activity consists of invoking an operation provided by a partner Web service, we identified the location identifiers by pairs of resources and actions described in the XACML policy. For example, the RefundItems operation offered by the ProductsWS Web service represents a location identifier. The generated aspect code integrates the access control verification before invoking Web service operations that require authorisation. It begins by invoking getUserAuthorization operation that passes the user's current role by parameter. If the user has the access rights, then the BPEL process continues by invoking the appropriate Web service operation. Otherwise, it returns an error message and the BPEL process exists.

Figure 10 Excerpt of generated BPEL aspect for access control

```
Aspect AccessControl
BeginAspect

Before
Invoke <OrderItem>
BeginBehavior
  <!-Accessed by the manager-->
  ...
EndBehavior

Before
Invoke <ApplyDiscount>
BeginBehavior
  <!-Accessed by the manager and supervisor-->
  ...
EndBehavior

Before
Invoke <RefundItems>
BeginBehavior
<!-Initialize Authentication Request Message-->
<bpel:assign validate="no" name="Assign parameters to Authorization message">
   ...
<!-Invoking the Authorization WS-->
<bpel:invoke name="Invoke Authorization" partnerLink="Authorization" operation="
    getuserAuthorization" portType="ns:Authorization" inputVariable="
    AuthorizationRequest" outputVariable="AuthorizationResponse">
</bpel:invoke>

  <!-- if User doesn't have access rights, Reply with error message and Exit
      Process-->
<bpel:if name="If Permission denied">
   <bpel:condition><![CDATA[($AuthorizationResponse.parameters/ns:
       userAuthorizationReturn!=True)]]>
</bpel:condition>
<bpel:sequence>

<!-Initialize the ErrorString message-->
...
<!-Copy the AuthorizationResponse to the ErrorStr -->
...
<!-Invoke GetErrorStr WS-->
...
<!-Initialize Bpel Output Message-->
...
<!-Copy Authentication ErrorStr to Output variable-->
...
<!-Return "User Not Authorized" to the Client"-->
<bpel:reply name=" Return ErrorString" partnerLink="client" operation="process"
    portType="tns:AnyWSProcess" variable="output">
</bpel:reply>
</bpel:sequence>
</bpel:if>

EndBehavior

Before
Invoke <BuyItems>
BeginBehavior
  <!-Accessed by the manager, supervisor, employee and client-->
  ...
EndBehavior

Before
Invoke <RequestShipment>
BeginBehavior
  <!-Accessed by the manager, supervisor, employee and client-->
  ...
EndBehavior
EndAspect
```

Figure 11 OPS secure BPEL process (see online version for colours)



6.3 Experimental results

Weaving the security aspects in Figure 9 and Figure 10 within the BPEL process of OPS presented in Figure 2 produces the secure BPEL process illustrated in Figure 11. The resulted BPEL process provides dynamic authentication and role based access control features for OPS. The BPEL process begins by receiving a request to a service from a user. After receiving the input, the authentication Web service gets invoked and the user will be asked to enter his credentials to ensure that the user has valid credentials. 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 authentication Web service will return the current user's role and the process will continue to service the client's request. When the process reaches an invoke activity that requires access control (as indicated in the process's security policies), 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 AnyOPSWebservice operation and return the client's requested service.

Verifying the successful integration of the RBAC-OPS security features in the original BPEL code of OPS 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. Note that these security AspectBPEL aspects are invoking security Web services, that we will refer to as "Security Enforcers". These security enforcers are ensuring authentication and role based access control for the OPS process. We assume that the OPS administrator will import the appropriate WSDL file of the security enforcer services. In our future work, we aim to elaborate a methodology for the "Security Enforcers" that will be built on standards for WS security, and will allow centralising security at the BPEL level, while ensuring both dynamic adaptation for changes in the security requirements and minimising overheads imposed by invoking security checks at each Web service invoke.

7 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 demonstrate the scalability of our approach. The execution time has been measured using The Eclipse Test and Performance Tools

Platform (TPTP) (TPTPEclipse, see www.eclipse.org/tptp/). 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 12 (a) Performance analysis on the execution, (b) compilation and (c) weaving time of the AspectBPEL Framework (see online version for colours)

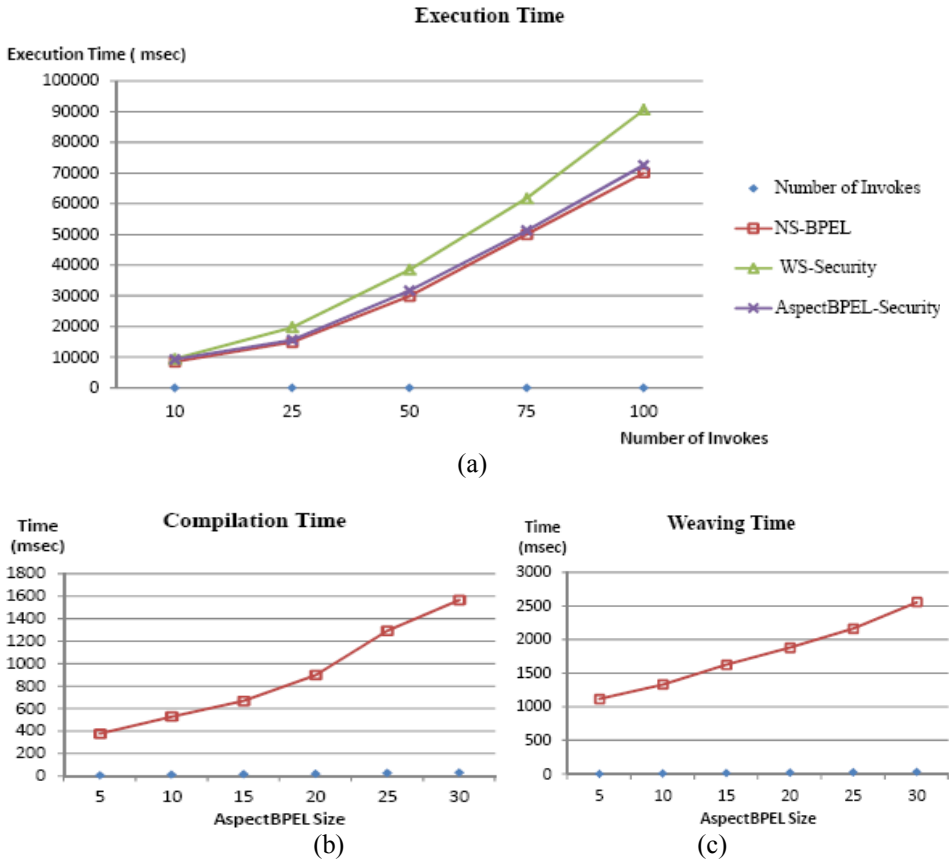


Figure 12a illustrates the following results:

- NS-BPEL process runs at an average of 8493 msec when enclosing 10 Web services invokes. It augments linearly to an average of 70,000 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 90,000 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 72,000 msec for 100 Web services invokes. This gives a redeem of more than 10,000 msec from the WS-Security process. The achieved results are promising and so close to the NS-BPEL process ones.

Next, we measured the compilation and weaving time of the AspectBPELSecurity framework using the EclipseTPTP. The compilation and weaving runtime represent the time needed to weave the AspectBPEL security aspects into a BPEL process. By looking at Figure 12b and Figure 12c, 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/advices.

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 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 *compilation time + weaving time + runtime of the process* = 1295.53 + 2165.211 + 15687.758 = 19147 < 19737 *(runtime of WS-Security)*.

Finally, Table 2 shows how the size of the *.bpel* and *.wsdl* files of the OPS 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, we notice that the BPEL file requires additional 94 lines of code to secure a BPEL process orchestrating 3 Web services. Here, the benefit of AOP prevails again since it allows dynamic and automatic weaving of these security activities without the need for manual work.

Table 2 Size of the OPSProcess BPEL and WSDL files

	NS-BPEL	*WS-Security*	*AspectBPEL-Security*
OPSProcess.bpel	178 Lines	264 Lines	272 Lines
OPSProcess.wsdl	67 Lines	80 Lines	76 Lines

8 Related work

In this section, we provide an overview on the related work in the area of Web services security, and the use of AOP to allow modularity for modelling 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 ideas, 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) (Lockhart, 2008), WSSecurity (Atkinson, 2006) and WS-XACML (Moses, 2011) were proposed.

SAML (Lockhart, 2008) is a specification language that is proposed by OASIS. Based on XML, it is used to specify security credentials, which are expressed as assertions. SAML can be used to manage secure sessions between organisations 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 (Atkinson, 2006) 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 authorisation of SOAP messages.

The Web Service eXtensible Access Control Markup Language (WS-XACML) (Moses, 2011) is proposed by OASIS as XML-based language to specify and exchange access control policies. WS-XACML is designed to define authorisation policies for principals that are specified using XML.

Bhatti et al. (2003) proposed X-RBAC: a 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.

Agostino Ardagna et al. (2006) proposed a design of a Web service architecture for enforcing access control policies. They also provided an example of implementation based on the WS-Policy (Schlimmer, 2004; Nolan, 2004) 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 realises 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.

Paci et al. (2008) proposed a RBAC-WS-BPEL framework for defining authorisation policies and constraints for WS-BPEL business processes. WS-BPEL is a language for composing Web services. To specify authorisation policies, the authors used XACML. They introduce the Business Process Constraint Language (BPCL), which can be used to specify authorisation 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 such as 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 and Mezini (2004) introduced a tool called AO4BPEL, which 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 (Di Francescomarino and Tonella, 2009). An aspect code is developed for a specific BPEL process, which makes it difficult to reuse. Co-AOP alleviates that challenge by introducing what is called the Explicit Join Points (EJP). These EJPs allow the base code to be aware of the aspect interfaces, and hence improve aspect reusability by decoupling base code and aspects. The aspects are initiated in the base code and described in their advices code, which forces the communication to be parameterised between both the base and aspects codes. 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 induces some performance 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, which are associated with the aspect code.

Hummer et al. (2011) introduced an integrated approach for identity and access management (IAM) in a SOA Context. Their approach is based on the elaboration of a domain specific language (DSL), to define an IAM policy that enforces role-based access control security for SOAs. The main contribution of their work is specifying the RBAC permission with regards to a certain context, and matching each context element to a WS-BPEL scope element. This context element allows single-sign on by reusing the same SAML assertion for each activity within a single scope. With every scope change, a new SAML assertion is generated for the corresponding role and context. However, our approach adopts an aspect oriented mechanism to weave security aspects in a WS-BPEL at any WS-BPEL element, and is not restricted to a scope element. Also, our approach offers the ability to dynamically update the WS-BPEL composition at run-time to adapt to new security requirements and separation of concerns between security and business logic.

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 (Shah, 2003), which is a superset of the C programming language to integrate security in C programs. De Win, in his Ph.D. thesis (DeWin, 2004), also discussed an AOP approach that adopts the concept of AOSD for defining security aspects to be weaved within applications. Bodkin (2004) presented a survey on the various security requirements within enterprise applications, with a focus on security cross cutting concerns particularly in authentication and authorisation. Furthermore, we find the Java Security Aspect Library (JSAL), in which Huang et al. (2004) 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.

Slowikowski and Ziekinski (2003) discussed some security solutions implemented in AspectJ that combines J2EE, JBoss application server, Java Authentication and Authorisation service API (JAAS) and Resource Access Decision Facility (RAD). These approaches are useful to explore the feasibility of using AOP for hardening in general software security.

9 Conclusion

We presented in this paper a new approach for the dynamic enforcement of Web services security. Our approach 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. Also, it 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 RBAC-OPS model and their BPEL aspects, deploying them dynamically in the BPEL process of OPS and the performance analysis demonstrate the feasibility and appropriateness of our propositions. Moreover, they illustrate the successful dynamic integration and modification of authentication and access control features in OPS. Our future works includes the extension of the proposed approach to deal with other concerns such as performance and also the verification of the correctness of the AspectsBPEL code.

Acknowledgements

This work is supported by the Lebanese American University (LAU) and CNRS, Lebanon.

References

Agostino Ardagna, C., Damiani, E., De Capitani di Vimercati, S. and Samarati, P. (2006) 'A web service architecture for enforcing access control policies', *Electronic Notes Theoretical Computer Science*, Vol. 142, p.4762.

Atkinson, B. (2006) *Web services security (WS-Security)*. Available online at: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

Bhatti, R., Joshi, J., Bertino, E. and Ghafoor, A. (2003) 'Access control in dynamic XML-based web-services with X-RBAC', *Proceedings of the International Conference on Web Services (ICWS 03)*.

Bodkin, R. (2004) 'Enterprise security aspects', Proceedings of the AOSD 04 Workshop on AOSD Technology for Application-level Security.

Charfi, A. and Mezini, M. (2004) 'Aspect-oriented web service composition with AO4BPEL', *ECOWS 04*.

DeWin, B. (2004) Engineering Application Level Security through Aspect Oriented Software Development*, PhD Thesis, Katholieke Universiteit Leuven.*

Di Francescomarino, C and Tonella, P. (2009) 'Cooperative aspect oriented programming for executable business processes', *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems*, Vancouver, Canada.

Evermann, J. (2007) 'A meta-level specification and profile for AspectJ in UML', *Journal of Object Technology*, Vol. 6, No. 7, pp.27–49.

Fuentes, L. and Sanchez, P. (2006) 'Elaborating UML 2.0 Profiles for AO Design', *Proceedings of the International Workshop on Aspect-Oriented Modeling*.

Huang, M., Wang, C. and Zhang, L. (2004) 'Toward a reusable and generic security aspect library', *Proceedings of the AOSD'04 Workshop on AOSD Technology for Application level Security*.

Hummer, W., Gaubatz, P., Strembeck, M., Zdun, U. and Dustdar, S. (2011) 'An integrated approach for identity and access management in a SOA context', *Proceedings of the 16th ACM symposium on Access Control Models and Technologies (SACMAT 11)*, New York, USA.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G. (2001) 'An overview of AspectJ', *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP 01)*, Springer-Verlag, London, UK.

Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, Ch., Lopes, C., Loingtier, J-M. and Irwin, J. (1997) 'Aspect-oriented programming', in Aksit, M. and Matsuoka, S. (Eds): *Proceedings European Conference on Object-Oriented Programming*, Springer-Verlag, Berlin, Heidelberg.

Lockhart, B. (2008) *OASIS Security Services TC (SAML)*. Available online at: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

Moses, T. (2011) *OASIS eXtensible Access Control Markup Languag e(XACML), OASIS Standard 2.0*. Available online at: http://www.oasis-open.org/committees/xacml/

Nolan, P. (2004) *Understand WS-Policy processing*, Technical report, IBM Corporation.

Paci, F., Bertino, E. and Crampton, J. (2008) 'An access-control framework for WS-BPEL', *International Journal of Web Services Research*, Vol. 5, No. 3, pp.20–43.

Pavlich-Mariscal, J., Michel, L. and Demurjian, S. (2007) 'Enhancing UML to model custom security aspects', *Proceedings of the 11th International Workshop on Aspect- Oriented Modeling*.

Schlimmer, J. (2004) *Web Services Policy Framework (WS-Policy)*. Available online at: http://www-128. ibm.com/developerworks/Webservices/library/specification/ws-polfram/

Shah, V. (2003) *An Aspect-Oriented Security Assurance Solution*, Technical Report AFRL-IF-RS-TR-2003-254, Cigital Labs.

Slowikowski, P. and Zielinski, K. (2003) 'Comparison study of aspect-oriented and container managed security', *Proceedings of the ECCOP Workshop on Analysis of Aspect-Oriented Software*.

Sun, M., Li, B. and Zhang, P. (2009) 'Monitoring BPEL-based web service composition using AOP', *Proceedings of The 8th IEEE/ACIS International Conference on Computer and Information Science*, Washington, DC, USA.

Wu-Lee, C. and Hwang, G. (2010) 'Dynamic policies for supporting quality of service in service-oriented architecture', *Proceedings of the International Conference on Electronics and Information Engineering*, Washington, DC, USA.