Towards a BPEL Model-Driven Approach for Web Services Security

Hanine Tout[a], Azzam Mourad[a], Hamdi Yahyaoui[b], Chamseddine Talhi[c], Hadi Otrok[d]
[a]Department of Computer Science and Mathematics,
Lebanese American University, Beirut, Lebanon
[b]Computer Science Department, Kuwait University, Kuwait
[c]Computer Science Department, ETS Montreal, Canada
[d]ECE Department, Khalifa University of Science, Technology & Research, UAE

***Abstract*—By handling the orchestration, composition and interaction of Web services, the Business Process Execution Language (BPEL) has gained tremendous interest. However, such process-based language does not assure a secure environment for Web services composition. The key solution cannot be seen as a simple embed of security properties in the source code of the business logic since the dynamism of the BPEL process will be affected when the security measures get updated. In this context, several approaches have emerged to tackle such issue by offering the ability to specify the security properties independently from the business logic based on policy languages. Nevertheless, these languages are complex, verbose and require programming expertise. Owing to these difficulties, specifying and the enforcing BPEL security policies become very tedious tasks. To mitigate these challenges, we propose in this paper, a novel approach that takes advantage of both the Unified Modeling Language (UML) and the Aspect Oriented Paradigm (AOP). By elaborating a UML extension mechanism, called UML Profile, our approach provides the users with model-based capabilities to specify aspects that enforce the required security policies. On the other hand, it offers a high level of flexibility when enforcing security hardening solutions in the BPEL process by exploiting the AOP approach. We illustrate our approach through an example of the dynamic generation and integration of model-based security aspects in a BPEL process.**
Keywords. Web Services, BPEL, Security, AOP, UML.

I. Introduction

With the drastic success of service-oriented computing, a wave of innovations has been unleashed to enhance the interaction between customers and service providers. Particularly, Web services composition is currently emerging as a convenient mechanism for automated interaction between distributed applications. BPEL (Business Process Execution Language) is so far the most important standard language for Web services composition. However, Web services composition via a BPEL process is still weak at the security level. The common practice used to harden the security of a BPEL process is to embed the security verification code within the the business logic of such process. However, this practice has several shortcomings. One of them is the negative impact that will be spawned at the business process deployment level. In fact, with the continuous growth of security threats and attacks, new countermeasures have to be released accordingly. Hence, the security measures embedded inside the code will be frequently updated. More specifically, when any of the security measures changes, the user should stop the process deployment, make the needed modifications and then redeploy it making all the offered services unavailable during the update procedure. This lack of inability to perform a dynamic update of the security requirements certainly affects the performance of the BPEL process. In this context, some dynamic approaches were proposed to enforce Web services security. They are based on policy languages like SAML (Security Assertion Markup Language) [1], WS-Security [2] and XACML (eXtensible Access Control Markup Language) [3]. However, policy languages are verbose, complex and require expertise. In other words, using such languages, security properties specification cannot be done without typically having a good programming experience.

To cope with the aforementioned problems, we propose in this paper a new approach that takes advantage of the Unified Modeling Language (UML), the Aspect Oriented paradigm and its supporting methods and tools at both design and code levels. Our proposal leverages the UML Profile concept, which is one of the UML extension mechanisms that allows UML models to be customized for specific domains. We adopt this model-based approach as an alternative for policy languages that require a high level of expertise for the advantages it offers. For instance, the security designer has to acquire only model-based capabilities for specifying BPEL security requirements. Moreover, modeling security properties based on graphical notations is less complex/error prone than developing them using policy languages.

Besides the benefits of using UML to model security requirements, we advocate for the separation of concerns earlier in the system. This provides the user with the ability to avoid the huge cost of planning the security as an afterthought phase. The separation of concerns is based on the use of the AspectBPEL language [4] that is built on top of AOP. The language constructs introduced to specify security requirements are aspect oriented. This allows a natural refinement of the AspectBPEL language [4] to be an end-to-end aspect oriented

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

driven model for security hardening in the context of Web services engineering. By using aspects, AspectBPEL provides a modularity for modeling cross-cutting concerns. It also offers the ability to specify some joint points where the AspectBPEL security aspects can be dynamically activated in the BPEL process. The main contributions of our approach are threefold:

1) The proposal of a meta-language to define BPEL security aspects at the design level.
2) The implementation of the proposed meta-language using a UML profile.
3) The design and implementation of a parser that allows the generation of an AspectBPEL code from the design.

We demonstrate the usefulness of our approach through a case study related to a Travel Agency System (TAS). First, using the IBM BPEL-UMLProfile, we designed a BPEL process of composite Web services for TAS. Then, using our AspectBPEL-UMLProfile, we designed the AspectBPEL aspect that captures the security requirements needed in such system. Its corresponding AspectBPEL aspect code is automatically generated using our parser. Finally, thanks to the platform that we have developed, we have been able to weave the generated BPEL security aspects within the BPEL process.

The remainder of this paper is organized as follows. In Section II, we present some notations that will be used in the rest of the paper to make it easy to understand. Section III, provides an illustrative example. Section IV is devoted to the description of the proposed approach. A case study is given in Section V to illustrate our proposal and support our claims. In Section VI, we discuss the related work. Finally, we conclude the paper and present our future work in Section VII.

II. Background

Before introducing our approach, we provide two short sections to make the understanding of our approach easier. The first one recalls the main definitions of the AspectBPEL language while the second, contains a brief description about IBM BPEL-UMLProfile.

A. AspectBPEL Overview

AspectBPEL is an Aspect-Oriented language for BPEL that is built on top of the AOP paradigm. It allows the definition of BPEL security aspects through notations close to those of AOP. Hereafter, we present its constructs:

a) BPEL_Aspect: Represents the BPEL security aspect. It is composed of the keyword `Aspect`, the aspect name and the `BPEL_Aspect_Body`. This latter implies the `BPEL_Location_Behavior` wrapped between `BeginAspect` and `EndAspect` keywords.

b) BPEL_Location_Behavior: Represents the advice-pointcut combination within an aspect. Each aspect may include one or many `BPEL_Location_Behavior` that is composed of the `BPEL_Insertion_Point`, `BPEL_Location_Identifier` and `BPEL_Behavior_Code`.

c) BPEL_Insertion_Point: Specifies the point where the code will be inserted with respect to the `BPEL_Location_Identifier`. It can have one of the following values: `Before`, `After` or `Replace`. The `Before` and `After` construct mean insert the new code before or after the identified location respectively. While the `Replace` means replace the old code at the stated location with the new code.

d) BPEL_Location_Identifier: Identifies the joint point or sets of joint points in the process 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 along with their signatures. These activities are divided into three categories: *Actions, Control* and *Fault*.

e) BPEL_Behavior_Code: Contains the code written in XPath language, that will be weaved to the BPEL Process. This code is wrapped between `BeginBehavior` and `EndBehavior` keywords. It will either be inserted before/after or replace the code at the location identifier previously stated.

B. IBM BPEL-UMLProfile

IBM offered a BPEL-UMLProfile [5] that allows the development of BPEL processes of composite Web services out of UML models. Their released mapping from UML to BPEL can be described as follows:

- **<<process>> class** corresponds to **BPEL process definition**. *The behavior of the process class is described using a UML activity diagram.*
- **Activity graph on a <<process>> class** corresponds to **BPEL activity hierarchy**.
- **<<process>> class attributes** corresponds to **BPEL variables**.
- **Hierarchical structure and control flow** corresponds to **BPEL sequence and flow activities**.
- **<<receive>>, <<reply>>, <<assign>>, <<invoke>> activities** corresponds to **BPEL activities**.

III. Illustrative Example

In this section, we describe the architecture of the Travel Agency System (TAS) and its corresponding BPEL process illustrated at both process and IBM-UMLProfile model levels.

A. Travel Agency System Overview

TAS is one of the most common Web services application. Therefore, we took it as a case study to demonstrate the usefulness of our approach. Figure 1 depicts the interaction between the user and TAS including its composition of Web services.

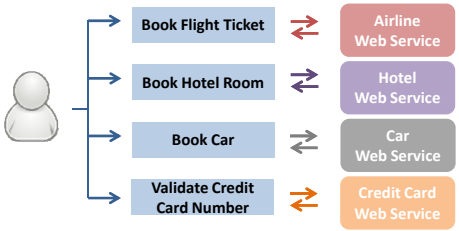

Figure 1. TAS Architecture

TAS consists of a BPEL process, its composition of four Web services and a graphical user interface that includes in its main page the services offered by this system. For instances, **Book Flight Ticket**, **Book Hotel Room**, **Book Car** and **Validate Credit Card Number**. The first service allows the user to book a ticket after consulting the **Airline Web Service**.

To reserve a room, the user can use the second service that includes a call for the **Hotel Web Service**. The user has also the ability to book a car from a **Car Web Service**. Finally, to validate the credit card number, a call for the **Credit Card Web Service** is needed.

B. BPEL Process Architecture

Figure 2 depicts only a part of the BPEL process architecture of TAS. It illustrates the latter at the process (a) and IBM-BPELUMLProfile level (b). It shows only one invoke activity that calls the appropriate TAS Web service operation. This activity is referred to as **InvokeAnyTASService**.

Upon the user request, the BPEL process gets invoked. It begins by assigning the user Request to AnyTASServiceRequest, then it calls the appropriate operation of the requested service and returns the Response to the client. As can be noticed, anyone may invoke this process since it lacks security measures.

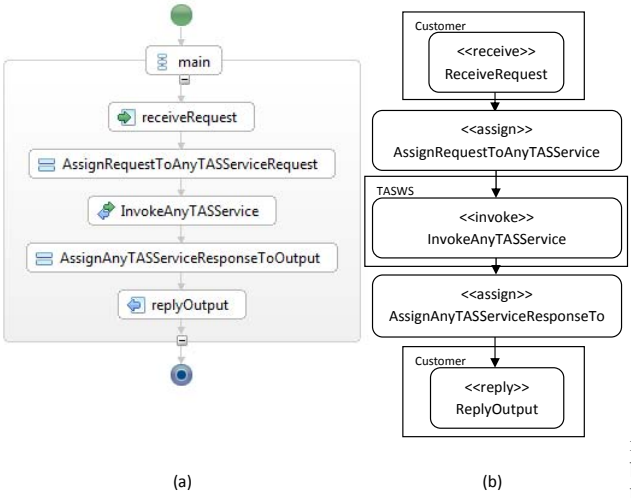

Figure 2. TAS BPEL at the Process and IBM-BPELUMLProfile Level respectively

IV. Main Approach

In this section, we present an overview of the proposed approach that aims to provide a model-driven engineering of BPEL-based secure composite Web services. Figure 3 depicts our approach schema by illustrating the interaction between its components. As shown in the figure, the security requirements are developed as separated entities and specified at the design level using the AspectBPEL-UMLProfile. In a previous work, IBM has developed a BPEL-UMLProfile that allows users to model BPEL processes and generate their corresponding code through an appropriate toolkit. we took advantage of their work to build our AspectBPEL-UMLProfile on top of the proposed BPEL-UMLProfile [5]. Our approach offers the user an abstract and user friendly environment to specify aspect-oriented security solutions.

The schema encompasses the following: (1) Developing the security requirements within a UML model based on the AspectBPEL-UMLProfile; (2) Applying the AspectBPEL-UMLProfile on a part of its elements (`BPEL_Aspect`, `BPEL_Location_Behavior`, `BPEL_Insertion_Point` and `BPEL_Location_Identifier` classes) and the BPEL-UMLProfile on the rest of it (`BPEL_Behavior_code` class); (3) Generating the aspect name, the insertion point and the location identifier from the first application (i.e., Applying the AspectBPEL-UMLProfile) and the BPEL code from the second application (i.e., Applying the BPEL-UMLProfile); (4) Generating automatically the complete AspectBPEL code using our UMLModelToAspectBPEL Converter; and (5) Conveying the generated code to the AspectBPEL weaver with the selected BPEL process in order to produce a secure BPEL process.

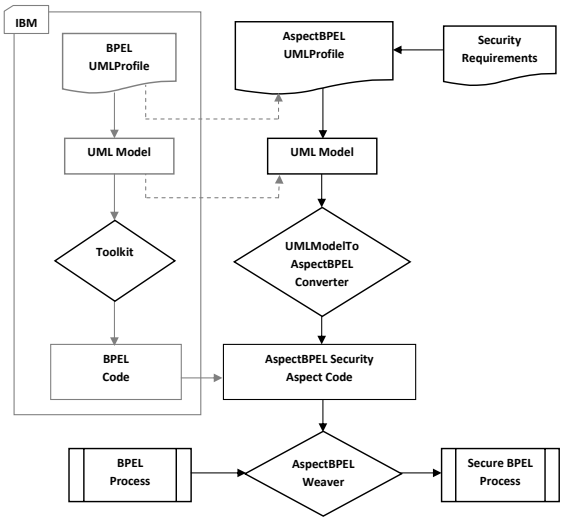

Figure 3. Approach Architecture

The components of our approach architecture are the following: (1) AspectBPEL-UMLProfile that offers the user a model-based technique to specify the security requirements to be hardened in the BPEL process; (2) UMLModelToAspectBPEL Converter that generates automatically the AspectBPEL aspects according to the AspectBPEL-UMLProfile; and (3) AspectBPEL Weaver that dynamically enforces security requirements into the BPEL process. Our approach spans over three phases. The first phase corresponds to security requirements specification. It allows to specify security concerns in separate components and generates automatically their corresponding aspects expressed in AspectBPEL [4]. In the second phase, AspectBPEL security aspects are generated using the developed converter. Finally, the third phase offers a tool for the dynamic weaving of the generated aspects within the BPEL process. In the sequel, we provide the technical details about each phase.

AspectBPEL-UMLProfile: We present in this section the meta-model specification of the AspectBPEL-UMLProfile. It captures the security properties based on the aspect concept offered by AspectBPEL. As depicted in Figure 4, the `BPEL_Aspect` stereotype extends the Metaclass `Class`. It is associated to one (or more) `BPEL_Location_Behavior` stereotype that is composed of two other stereotypes: `BPEL_Location_Identifier` and `BPEL_Behavior_Code`. However, these stereotypes could be associated with different `BPEL_Location_Behavior`. On the other hand, the `BPEL_Behavior_Code` element is related to the `Process`

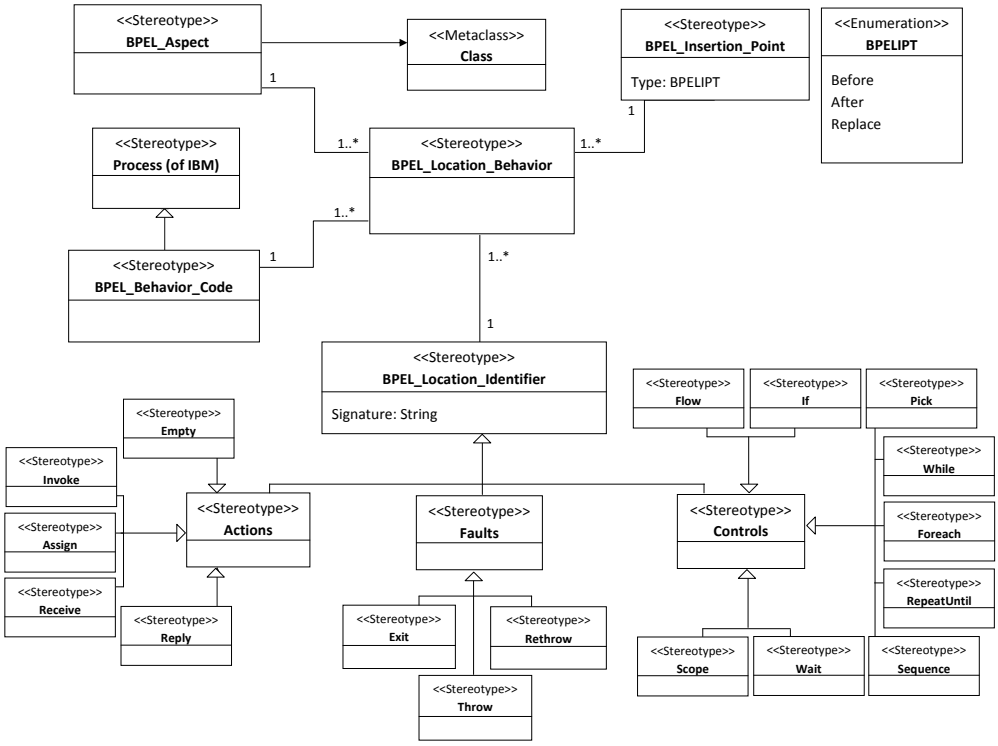

Figure 4. AspectBPEL-UMLProfile Meta-Model

stereotype offered by IBM. This is due to the fact that we use their BPEL-UMLProfile to express the `BPEL_Behavior_Code` element. This latter represents the code that will be integrated in the BPEL process. Activity stereotypes that are specialized from the `BPEL_Location_Identifier` (For instances, `Invoke`, `Assign`), are divided into three categories (e.g., Actions, Faults and Controls) and will be mapped to the BPEL process activities. They identify the joint point or sets of joint points in the process where the changes specified in the `BPEL_Behavior_Code` should be applied. As shown in the figure, this type of stereotypes has a tag definition called `Signature` of type `String`. This tag is unique in each location identifier and used to denote this latter. The last stereotype is `BPEL_Insertion_Point`. It identifies the point where the code should be injected depending on the value of its attribute `Type`. The values for `Type` are provided by the enumeration `BPELIPT`. With few UML knowledge, the user can model the security hardening solutions through this profile. It does not impose neither security expertise nor programming experience.

UMLModelToAspectBPEL Converter: Once the AspectBPEL-UMLProfile is applied on a UML model, the resulted design can be exported to an XML file using any software (In this work, VP-UML [6] is used) that supports this feature. Subsequently, the converter parses the XML file to automatically generate the AspectBPEL security aspect. This converter is based on the DOM parser for XML, and is written in Java. An example of the converter output is shown in Section V.

AspectBPEL Weaver: This section is devoted to present our AspectBPEL weaver for the AspectBPEL language. In a previous work [4], we have developed this weaver that allows the integration of the generated AspectBPEL aspect code within the specified BPEL process. It starts by compiling an AspectBPEL aspect, and then performing the needed weaving. More precisely, it fetches first the insertion point (i.e., Before, After or Replace). Then, it takes the location behavior, which represents the activity type and activity name where the behavioral BPEL code will be inserted.

We have integrated a BPEL parser based on the *ode.bpel* library [12] in the AspectBPEL framework in order to find the defined join points in the BPEL process. Once found, the behavioral code will be inserted in the original code and a message indicating that the weaving was successfully done will appear in the output console.

V. Case Study

This section illustrates the development of TAS as well as the mechanisms presented in our proposed approach for the dynamic enforcement of security solutions needed in such system. It demonstrates that our proposal requires only design skills to specify and enforce security solutions in a BPEL process. It shows also how this approach offers a graphical tool for modeling security requirements and hence how it can be a suitable alternative to the use of complex policy languages.

Seeing that it is the most advanced and mature language for Access Control specification, we opted to use XACML as representative for other policy languages. Listing 1 illustrates an XACML policy that permits the `manager` to invoke any

service called `InvokeAnyTASService` (Line 30 to Line 37) of any TAS Web Service (Line 22 to Line 29). This policy is used to check the role of the user (Line 5 to Line 12) and decide whether he has the right to access the service before the invoke of the stated action (e.g., InvokeAnyTASService). However, implementing, revising and changing XACML policies are all difficult, error-prone and time consuming. Moreover, hidden conflicts that may arise due to the diversity of roles in policies are difficult to locate and resolve.

Hence, we need a new solution that can easily check these errors and overcome these problems. Therefore, we present in this work a *Model-Driven Aspect-Oriented based Approach for Security Specification and Enforcement in BPEL*. The model-driven architecture for Web services security specification is easier to check and understand. Besides, the aspect-oriented paradigm offers a dynamic technique to enforce Web services security in the BPEL process.

Listing 1. Excerpt of TAS XACML Policy

```
[1]<PolicySet>
[2]<!--Role policy set for the Manager-->
[3]<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:
    schema:os" PolicyId="RPS:manager:role"
    PolicyCombiningAlgId="policy-combine:permit-
    overrides">
[4] <Target>
[5]  <Subjects>
[6]   <Subject>
[7]    <SubjectMatch MatchId="function:anyURI-equal">
[8]     <AttributeValue DataType="xml:anyURI">manager
    </AttributeValue>
[9]<SubjectAttributeDesignator AttributeId="role"
    DataType="xml:anyURI"/>
[10]   </SubjectMatch>
[11]  </Subject>
[12] </Subjects>
[13]</Target>
[14]<!--Include permissions of Manager role-->
[15]<PolicyIdReference>PPS:manager:role</
    PolicyIdReference>
[16]</Policy>
[17]<!--Permissions policy set for the Manager-->
[18]<Policy PolicyId="PPS:manager:role"
    RuleCombiningAlgId="rule-combine:permit-overrides
    ">
[19]<!--Permission to Invoke Any TAS Services-->
[20]<Rule RuleId="Permission:to:invoke:anytasservice"
    Effect="Permit">
[21]<Target>
[22] <Resources>
[23]  <Resource>
[24]   <ResourceMatch MatchId="function:string-equal">
[25]    <AttributeValue DataType="xml:string">AnyTASWS
    </AttributeValue>
[26]    <ResourceAttributeDesignator AttributeId="
    resource:resource-id" DataType="xml:string"/>
[27]  </ResourceMatch>
[28] </Resource>
[29]</Resources>
[30] <Actions>
[31]  <Action>
[32]   <ActionMatch MatchId="function:string-equal">
[33]    <AttributeValue DataType="xml:string">
    InvokeAnyTASService</AttributeValue>
[34]    <ActionAttributeDesignator AttributeId="action
    :action-id" DataType="xml:string"/>
[35]   </ActionMatch>
[36]  </Action>
[37] </Actions>
[38] </Target>
[39]</Rule>
[40]</PolicySet>
```

Last and not least, together a model-driven and aspect-oriented security hardening technique can realize more cost savings in terms of investment in time.

Hereafter, we present step by step how to apply our solution on TAS to enhance its security and performance.

- AspectBPEL Aspect Specification

In this step, the user designs a UML model that captures the access control property as well as the location where it should be applied. An example of the UML model is depicted in Figure 5. As shown in the figure, the BEPL aspect class is called `AccessControlAspect`. It contains the location where the access control will be applied. This location is called `AnyLocation`. This latter is composed of two elements. The first one is the location identifier that we call `AnyLocationIdentifier`. It will be mapped to a BPEL process activity. The second element is the BPEL process that will be invoked for access control obligation. In this example, we call this latter tt AccessControlProcess::AccessControl. The location is also linked to an insertion point called `AnyInsertionPoint`. It determines whether the BPEL code will be executed before, after or instead of the stated location identifier.

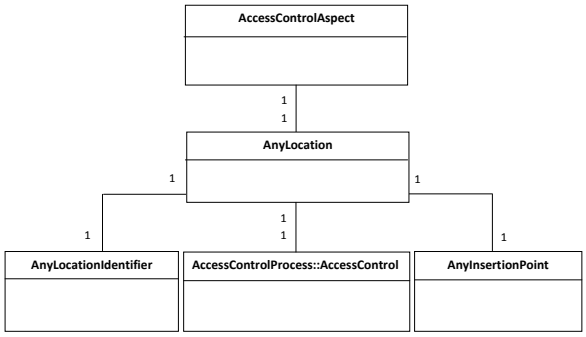

Figure 5. AspectBPEL Access Control Aspect UML Model

- Profiles Application

In this step, the user should apply the AspectBPEL-UMLProfile as well as the BPEL-UMLProfile on the model elaborated in the previous step. Technically, he must apply:

1) **BPEL_Aspect** stereotype on **AccessControlAspect**.
2) **BPEL_Location_Behavior** stereotype on **AnyLocation**.
3) **Invoke** or any of the **BPEL_Location_Identifier** stereotypes on **AnyLocationIdentifier**.
4) **BPEL_Insertion_Point** stereotype on **AnyInsertionPoint**.
5) **Process** stereotype on **AccessControlProcess::AccessControl**.

The first 4 stereotypes are described in the AspectBPEL-UMLProfile while the last one is offered by IBM in their BPEL-UMLProfile. The resulted model is shown in Figure 6.

Ought not to forget that the user has to specify the tag values of the attributes of each stereotype by going to its properties and writing the appropriate values (for instance `Signature=InvokeAnyService` and `Type=Before`). As illustrated in the figure, a UML class is used to model the access control process stereotyped with «`Process`», which

is a subset of the BPEL-UMLProfile [5]. On the other hand, its behavior is described in the activity diagram depicted in Figure 7. Upon receiving the user request, the login information is assigned to the Validator. Depending on the username and password entered by the user, the Validator deduces the role of the user. If this latter is deemed to be a manager, the access will be granted and `InvokeAnyTASService` of `AnyTASWS` will be executed. Otherwise, the access will be denied. When either the `InvokeAnyTASService` has been executed or the `Validator` has rejected, the appropriate information is returned to the user. The design of this security access control flow should be transformed into a code that can be integrated in the BPEL process to enforce its security. Here comes the role of aspect oriented programming.

Listing 2. Excerpt of Generated TAS Access Control Aspect

```
[1]Aspect AccessControlAspect
[2]BeginAspect
[3] Before
[4] Invoke <InvokeAnyTASService>
[5] BeginBehavior
[6]
[7]  ...
[8]
[9]  <invoke name="InvokeACCValidator" partner="
    AccessControlValidator" portType="asns:
    AccessValidatorPT" operation="check"
    inputVariable="loginDetails" outputVariable="role
    ">
[10]   <target linkName="assign-to-
    accesscontrolvalidator"/>
[11]  <source linkName="accvalidator-to-TASWS"
    transitionCondition="bpws:getVariableData('role',
     'value')='Manager'"/>
[12]  <source linkName="validator-to-setMessage"
    transitionCondition="bpws:getVariableData('role',
     'value')!='Manager'"/>
[13] </invoke>
[14]
[15] ...
[16]
[17] <assign name="AssignAccessDeniedToOutput">
[18]  <target linkName="accesscontrolvalidator-to-
    setMessage"/>
[19]  <source linkName="setMessage-to-reply"/>
[20]  <copy>
[21]   <from expression="'Denied'"/>
[22]   <to variable="replyInfo" part="Access"/>
[23]  </copy>
[24] </assign>
[25]
[26] ...
[27]
[28] <reply name="replyOutput" partner="Customer"
    portType="apns:CheckAccessPT" operation="
    sendReply" variable="replyInfo">
[29]  <target linkName="setMessage-to-reply"/>
[30]  <target linkName="TASWS-to-reply"/>
[31] </reply>
[32]
[33] ...
[34]
[35] EndBehavior
[36]EndAspect
```

- AspectBPEL Aspect Generation

This step is divided into three parts: (1) Generating a BPEL code from the activity diagram described above; (2) Generating an XML code from the rest of the model illustrated in Figure 6; and (3) Combining both codes to dynamically generate its corresponding AspectBPEL code using our UMLModel-ToAspectBPEL converter.

A synopsis of the generated AspectBPEL code is shown in Listing 2 (Due to space constraints, much of the detail is omitted). In Line 1, `AccessControlAspect` denotes the aspect name. The element `Before` (Line 3) represents the insertion point. `Invoke` represents the type of the BPEL activity and `InvokeAnyTASService` identifies its signature (Line 4). Finally, the code (Line 6 to Line 34) wrapped between `BeginBehavior` and `EndBehavior` represents the access control flow.

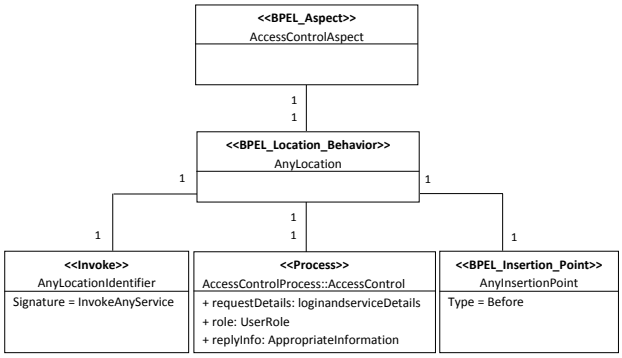

Figure 6. Applied Profiles on the UML Model

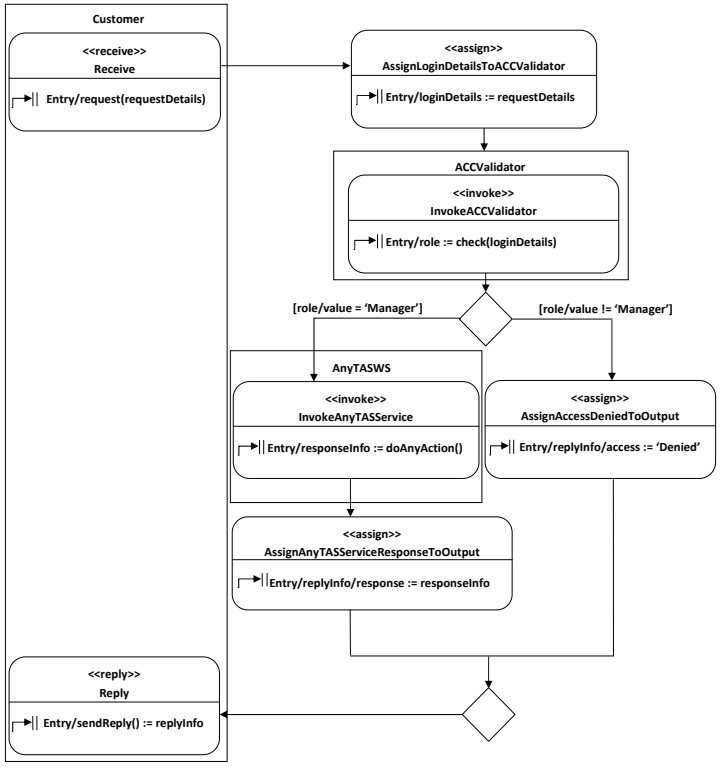

Figure 7. Activity Diagram for the Access Control Process

- AspectBPEL Security Aspect Weaving

The final step to be performed is to weave the generated AspectBPEL code into the TAS BPEL process. Using our AspectBPEL weaver, this step can be performed dynamically without the need to stop the deployment of the process. The weaving procedure consists in matching the combination of each insertion point, type of location identifier and its signature with the TAS BPEL activities. By doing so, the weaver is able to insert the access control code before the `InvokeAnyTASService` activity in the BPEL process.

Figure 8 reflects the modification at the BPEL process level after the weaving procedure. It shows that the access control verification code is integrated in the TAS BPEL process ensuring that only legitimate users can access the services offered by this latter. By comparing Figure 2 and Figure 8, you can notice that the secure BPEL process contains a call for a the Validator that returns the role of the user. Then, the If condition checks the returned value to decide whether the access should be granted or denied before the `InvokeTASAnyService` activity is executed. If the role returned by the Validator is equal to Manager, InvokeAnyService will be invoked and its response will be returned to the user. Otherwise, the process returns an Access Denied message and stops.

VI. Related Work

Few researches have been done in the area of generating BPEL code based on a model-driven approach. In [7], Li et al. proposed a new approach for modeling Web services composition using UML models then generating automatically their corresponding verified BPEL code. Taking advantage of an extended sequence diagram, they built an extended statechart diagram to model service composition. Then, using SMV (a model checking tool), they verified whether the behavior of Web services composition conducted from the statechart match the one described in the requirements specification. Finally, they generated the BPEL code from the verified models. However, in this approach, generating the input code of the SMV is not totally automatic. One part can be generated from the statechart diagram but the second has to be written manually. Furthermore, they did not explain neither present the technique used to generate the verified BPEL code. In a IBM initiative, Mantel [5], devised a UML profile to model BPEL processes and elaborated a mapping between the UML profile and the BPEL constructs. He also offered a tool to generate the BPEL code according to the UML models. In our work, we took advantage of their contribution to build the AspectBPEL-UMLProfile capturing the security properties.

On the other hand, some approaches were proposed to address the problem of modeling security properties such as access control properties. A UML based notation was proposed by Epstein and Sandhu [8] to model access control properties. With this notation, they were able to model the Role Based Access Control Framework for Network Enterprises. Practically, they exploited the stereotyping concept within UML to develop new classes and interfaces needed in such framework. However, their work did not cover the generation of a security code from the built models whereas in our work we are investigating the generation of an AspectBPEL security aspect code from a UML design model.

As for model-driven aspect-oriented approaches, there were also some interesting proposals. Following a bottom-up approach, Kande et al. [9] were able to model a code written in aspect-oriented programming language using UML. They started from the low level by writing the code of a logging aspect, then using the UML extension mechanism that is based on stereotypes, they created new model elements to form the design of such aspect needed in software design. Finally, they showed the configuration model before and after the weaving process. The main difference with our work is that we start from the design level to generate the AspectBPEL aspect code. In addition, in our approach, the weaving step is applied at the process level not at the UML model level. Groher and Schulze [10] presented a UML notation to model security aspects. They used the Together CASE tool to design this latter and then proposed a code generator to generate an AspectJ code from the security aspect design. However, their code generator was not accomplished. In contrast, we are offering a UMLModelToAspectBPEL converter that generates automatically the whole AspectBPEL aspect code according to the security requirements. A model driven code generation approach was proposed by Cooper et al. [11]. By extending the UML notation of the FDAF (Formal Design Analysis Framework), they modeled the RBAC aspect of an online Banking system. Then, they translated the design model to XML code and finally generated its corresponding AspectJ code. Yet, this latter does not contain the advice element of the aspect. Therefore, the user should write it manually to accomplish the implementation of the code. Conversely, in our work, the complete code can be generated using our UMLModelToAspectBPEL converter.

VII. Conclusion and Future Work

In this paper, we presented a model-driven and aspect oriented approach for security hardening of BPEL processes. We have shown how security solutions can be specified using the AspectBPEL-UMLProfile and integrated in the BPEL process through the AspectBPEL weaver. Our approach offers the ability to transform non functional requirements (e.g., security) from UML models into AspectBPEL security aspect specifications, and weaving them within the BPEL process for the sake of developing a secure composition of Web services. Our future work will focus on extending the AspectBPEL-UMLProfile to allow the specification of different and more sophisticated security properties.

References

[1] OASIS SAML TC. SAML. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security).

[2] OASIS WSS TC. Web services security (WS-Security). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

[3] T. Moses. OASIS eXtensible Access Control Markup Language(XACML), OASIS Standard 2.0. http://www.oasis-open.org/committees/xacml/.

[4] A. Mourad, S. Ayoubi, H. Yahyaoui, and H. Otrok. New Approach for the Dynamic Enforcement of Web Services Security. *In Proceedings of Privacy, Security and Trust Conference, PST 2010*, Ottawa, Canda, 2010.

[5] K. Mantell. From UML to BPEL. https://www.ibm.com/developerworks/webservices/library/ws-uml2bpel/.

[6] Visual Paradigm for UML 8.3. http://www.visual-paradigm.com/product/vpuml/.

[7] B. Li, Y. Zhou, and J. Pang. Model-driven Automatic Generation of Verified BPEL Code for Web Service Composition. *16th Asia-Pacific Software Engineering Conference*, Penang, Malaysia, 2009.

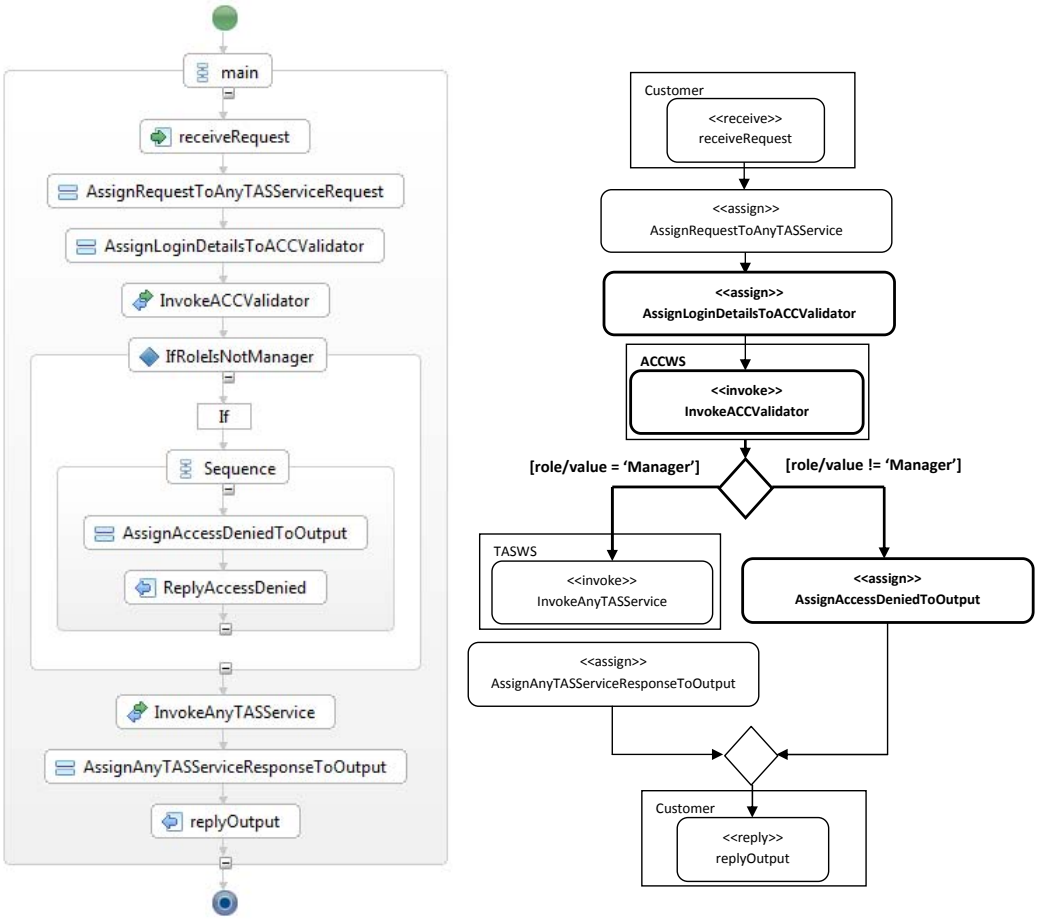

Figure 8. Secure TAS BPEL at the Process and IBM-BPELUMLProfile Level respectively

[8] P. Epstein and R. Sandhu. Towards A UML Based Approach to Role Engineering. *In Proceedings of the fourth ACM workshop on Role-based access control*, New York, USA, 1999.

[9] M. Kandé, J. Kienzle, and A. Strohmeier. From AOP to UML: Towards an Aspect-Oriented Architectural Modeling Approach. *Technical Reports in Computer and Communication Sciences, Faculté I&C, Ecole Polytechnique Fédérale de Lausanne*, Switzerland, 2002.

[10] I. Groher and S. Schulze. Generating Aspect Code from UML Models. *Workshop on Aspect-Oriented Modeling with UML, AOSD*, Boston, USA, 2003.

[11] K. Cooper, L. Dai, S. Dascalu, N. Mehta, and S. Velagapudi. Towards Aspect-oriented Model-driven Code Generation in the Formal Design Analysis Framework. *International Conference on Software Engineering Research and Practice (SERP'07)*, 2007.

[12] Jarvana Search. Overview about Apache ode. http://www.jarvana.com/jarvana/view/org/apache/ode/apache-ode-docs/1.2/apache-ode-docs-1.2.zip!/javadoc/overview-summary.html.