Incorporating Security Requirements into Service Composition: From Modelling to Execution

Andre R.R. Souza[1], Bruno L.B. Silva[1], Fernando A.A. Lins[1], Julio C. Damasceno[1], Nelson S. Rosa[1], Paulo R.M. Maciel[1], Robson W.A. Medeiros[1], Bryan Stephenson[2], Hamid R. Motahari-Nezhad[2], Jun Li[2], and Caio Northfleet[3]

[1] Federal University of Pernambuco, Centre of Informatics
{arss,blbs,faal2,jcd,nsr,prmm,rwam}@cin.ufpe.br
[2] HP Labs Palo Alto
{bryan.stephenson,hamid.motahari,jun.li}@hp.com
[3] HP Brazil
caio.northfleet@hp.com

Abstract. Despite an increasing need for considering security requirements in service composition, the incorporation of security requirements into service composition is still a challenge for many reasons: no clear identification of security requirements for composition, absence of notations to express them, difficulty in integrating them into the business processes, complexity of mapping them into security mechanisms, and the complexity inherent to specify and enforce complex security requirements. We identify security requirements for service composition and define notations to express them at different levels of abstraction. We present a novel approach consisting of a methodology, called Sec-MoSC, to incorporate security requirements into service composition, map security requirements into enforceable mechanisms, and support execution. We have implemented this approach in a prototype tool by extending BPMN notation and building on an existing BPMN editor, BPEL engine and Apache Rampart. We showcase an illustrative application of the Sec-MoSC toolset.

1 Introduction

There is an increasing need for considering security requirements in service composition. Service providers and service consumers need security guarantees to offer and use services. The users of composite services have requirements such as the confidentiality of their information being preserved by all service providers participating in the composition. Service providers may also sign SLAs (Service Level Agreements) with the service customers, which impose security requirements that must be satisfied when the service is being delivered.

Despite the importance, the incorporation of security requirements in service composition development is still a challenge for many reasons: no clear identification of security requirements for service composition, absence of notations to express them, difficulty in integrating them into the business process (behind the service composition), complexity of mapping them into actual security mechanisms, lack of monitoring

L. Baresi, C.-H. Chi, and J. Suzuki (Eds.): ICSOC-ServiceWave 2009, LNCS 5900, pp. 373–388, 2009.

mechanisms to check whether they are being satisfied at execution time and the complexity inherent to specify and enforce security requirements.

Current solutions on service composition and security usually concentrate on a particular aspect: incorporation of security requirements into the business process definition [10][14][11][12], executable composition [3][18][8][5][6] or enforcement of security at execution time [10][16]. However, there is little work that supports identification, expression and enforcement of security requirements for service composition at all levels of abstraction.

In this paper, we present an approach to deal with security requirements of service composition at various levels of abstraction. Our approach streamlines the modelling and enforcement of security requirements across multiple stages of business service development: from business process specification, to composite service design and development, and to business process execution.

We identify a collection of security requirements for service composition and propose a set of abstractions to express these security requirements. At the business level, we propose abstractions to express identified security requirements, which are represented by extending BPMN notations [19]. We present a methodology called Sec-MoSC (Security for Model-oriented Service Composition) through which composition logic, which is represented in standard BPMN, is annotated with security abstractions. The annotated model is translated into a BPEL process specification and configurations for security enforcement modules and mechanisms. The security requirements are enforced at runtime by an auxiliary engine, which complements the BPEL execution engine. We have implemented the approach in a prototype tool to demonstrate its viability and showcase it using an example.

This paper is structured as follows. Section 2 introduces an example scenario used throughout the paper. Section 3 presents security requirements and abstractions used to express them. Section 4 presents the Sec-MoSC methodology. Section 5 presents the architecture and implementation of the prototype tool and an example. Section 6 presents related works. Section 7 draws our conclusions and identifies future work.

2 Illustrative Scenario

We consider a virtual travel agency called VTA (Virtual Travel Agency), which provides services through an Internet portal for trip arrangements. The VTA portal is an interface between many different travel business companies/government services and end-users interested in travel.

Customers interact with VTA for service usage, payment and non-computational assets (e.g., receive the travel insurance policy). The operations are accessed through a Web interface available in the VTA portal. The VTA portal runs though the composition of services available in the Internet. We identified important security requirements including (i) encrypting credit card information in all communications, (ii) VTA and its partners need valid digital signatures, (iii) authentication mechanisms must be used in all interactions among web services accessed by VTA, (iv) logging all operations in the service composition for auditing purposes, and (v) VTA and its partners require that their Web services may only accept requests coming from specific IP addresses or domains.

3 Security Abstractions for Business Modelling to Execution

Our approach starts with the identification of security requirements that may be present in service composition. In the following, we identify these requirements and introduce a set of abstractions (modelling elements) that help understand, express and structure them in such a way that they can be incorporated into a service composition.

3.1 Security Requirements

By examining the VTA and other service applications that require composition of multiple Internet services across different service providers, we have identified the following common security requirements:

[NFR 01] Confidentiality. Critical information stored or exchanged in service interactions, such as credit card and personal identification information in the VTA example, needs to be disclosed only as needed to deliver the service.

[NFR 02] Data Retention. The information exchanged between services may have a time-to-live (TTL), i.e. it should be deleted after a certain time. For example, VTA must delete credit card information after sending the e-ticket to the customer.

[NFR 03] Access Control. Service providers must guarantee that only employees of the commercial department have access to customer data (e.g., the system designer cannot access this data).

[NFR 04] Authentication. Authentication ensures that only appropriate users have access to the sensitive or critical information held by the services, and also ensures that an action X was performed by a user Y and by nobody else (non-repudiation).

[NFR 05] Restricted Access. The composite service may filter communication with other services based on IP address or domain names. The VTA service relies on the IP addresses belonging to registered service providers to restrict access.

[NFR 06] Data Integrity. Stored and communicated sensitive customer data (such as credit card information) has to be checked against data corruption.

[NFR 07] Data Sharing. Service providers may be required to keep customer data within their company, or share it with sub-contractors in a limited way.

[NFR 08] Service Certification. Services used in the composition may need a certified abstraction of their behaviour. Any service whose behaviour is altered must be certified again. A Secure Capability Authority may be in charge of evaluating the compliance of Web services with respect to security capabilities and functionality.

[NFR 09] Auditing. Any operation performed by the service composition may need to be logged for auditing purposes.

[NFR 10] Monitoring. Anomalous behaviour in the composition that leads to violation of security constraints (e.g. leakage of information) should be monitored.

This set of security requirements has served as a basis for the proposed solutions presented in this paper.

3.2 Abstractions

In order to express the security requirements, we propose a set of abstractions by extending our previous work [15] to express non-functional requirements at modelling, development and run time: *NF-Attributes*, *NF-Statements*, and *NF-Actions*.

Table 1. Security requirements, NF-Actions and properties

Security requirement (Section 3.1)	**NF-Action**	**Properties**
NFR01	*UseCryptography*	Encryption Type (Symmetric / Asymmetric), Algorithm, Encrypted Message Parts , Key Length
NFR02	*DeleteInformation*	Type (Time/Event based), Time-to-live (TTL), Event (Before, After)
NFR03	*UseAccessControl*	Role, list of trusted entities, access level for each service
NFR04	*UseAuthentication* *UseDigitalSignatures*	Token Type (Username / X509 / Kerberos / SAML), Session Timeout
NFR05	*RestrictAccess*	Restriction Type (allow, deny), Source IP, Source IP Range, Destination Endpoint
NFR06	*CheckDataIntegrity*	Signature Type (HMAC_SHA1 / RSA_SHA1), Checked Parts (Header / Body)
NFR07	*ClassifyInformation*	Classification (Top Secret, Secret, Confidential)
NFR08	*CertifiedServiceBehavior*	Level of certification, The trusted entity
NFR09	*Log*	Level (DEBUG, INFO, WARN, ERROR)

NF-Attribute. It models non-functional characteristics such as *security* that may be defined in service composition. An NF-Attribute can be primitive or composite. A composite NF-Attribute is decomposed into primitive NF-Attributes that are closer to implementation elements. The composite NF-Attribute *Security* may be decomposed into primitive NF-Attributes *Integrity* and *Confidentiality*.

NF-Action. It models a software aspect or hardware feature that implements an NF-Attribute. Software aspects mean design decisions, algorithms, data structures, configurations and so on. Hardware features concern computer resources available for running the software system. NF-Actions are the abstractions to express the security enforcement mechanisms that must be implemented to achieve the NF-Attribute. For example, the realisation of the NF-Attribute *Authentication* may be carried out by implementing the NF-Action *UseAuthentication*. Finally, NF-Actions may be grouped

to facilitate their reuse. They may have a set of properties like a tuple <name, value> that help to better characterise and implement them, e.g., the NF-Action *UseCryptography* has the property <*encryption type*, symmetric>.

NF-Statement. It models constraints defined on an NF-Attribute to guide decisions taken to implement the NF-Attribute. In the context of security, NF-Statements are defined in multiple levels such as *high*, *medium*, *low*. For example, "*High Confidentiality*" may require choosing public key-based encryption algorithm, whereas "*Medium Confidentiality*" may require 128-bit AES encryption.

In this paper, our focus is on "security" (a composite NF-Attribute). We define a set of primitive security NF-Attributes (*Integrity*, *Confidentiality*, *Authentication*, *Authorization*, *Audit, etc.*) and a set of NF-Actions (together with their properties) that may be used to realise the NF-Attributes (see Table 1), and four constraint levels to the NF-Statements (*High, Medium*, *Low* and *Other*). The identified NF-Attributes are realised by implementing these NF-Actions through configuring orchestration engines using existing standards, defining and implementing security modules, and so on.

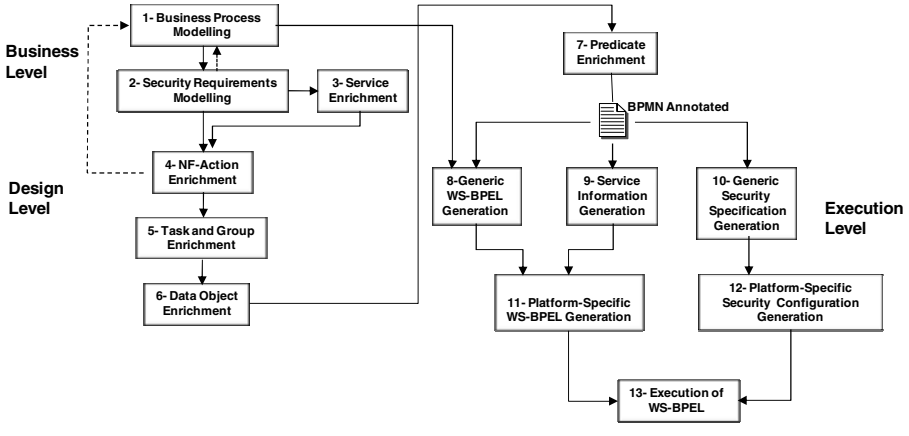

Fig. 1. Sec-MoSC Methodology

4 Sec-MoSC Methodology

We present a methodology, called Sec-MoSC, for incorporation of security requirements in service composition based on the following principles and assumptions: (i) different stakeholders (business users, security experts, service developers) are involved in defining and enforcing security requirements at various development stages. Following the principle of separation of concerns, we define three abstraction levels: *business*, *design* and *execution*. Security requirements are represented in different views corresponding to these layers; (ii) targeting business users, we adopt BPMN to express business processes; (iii) business users incorporate security requirements into the composition process during the business process definition using an extended BPMN notation; (iv) we use WS-BPEL to express the executable service composition

specification. We provide a semi-automated approach to translate the annotated BPMN into executable BPEL along with enforceable security mechanisms. The run-time support is offered by complementing the orchestration engine with an auxiliary engine. Sec-MoSC methodology, shown in Figure 1, consists of the following steps:

Step 1: Business Process Modelling: This step refers to modelling the business process using BPMN abstractions such as tasks, task groups, data objects (business process information), predicates and so on. This step is performed by a business expert that has knowledge of the business process.

Step 2: ***Security Requirements Modelling:*** This step consists of defining security requirements and binding them to elements of the BPMN model. We extend BPMN elements by three new notations corresponding to NF-Attribute, NF-Statement and NF-Action. Considering the non-functional abstractions introduced in Section 3.2, this step initially defines NF-Attributes (composite *security* or primitive *confidentiality*, etc.) and NF-Statements (*High, Medium*, *Low* and *Other)* and binds them to BPMN elements. This step is performed by a business expert, possibly aided by a security expert that knows the meaning of NF-Attributes and NF-Statements. The business expert may include the rationale to define the NF-Statement. NF-Attributes and NF-Statements are bound to a BPMN element like data object, task or task group. This step annotates the BPMN model with NF-Attributes, NF-Statements, and default NF-Actions. For example, a BPMN task may be associated to the NF-Attributes *Authorization* and *Authentication* and NF-Statement "*High*" along with default NF-Actions *RestrictAccess* and *UseAuthentication* needed to realise them.

Step 3: ***Service Enrichment:*** This step concentrates on enriching the annotated BPMN model by including additional information about the actual services that are used to realize BPMN tasks. Services may be selected from a service repository. A filter based on NF-Attributes and NF-Statements is applied to the list of candidate services in such a way that only services that satisfy the security requirements are considered. This step is to facilitate automated code generation from BPMN to BPEL.

Step 4: ***NF-Action Enrichment:*** This step further refines the security requirements identified in Step 2. The security expert considers if the set of NF-Attributes and default NF-Actions are enough to realise the security requirements. The security expert may change (remove/add/alter) the default set of NF-Actions and their properties to satisfy the requested NF-Statement. Each NF-Action has a particular set of properties that serves as parameters in its use (see Table 1). These properties may be altered by the security expert to select the best parameters to meet NF-Statements.

Steps 5, 6 and 7: ***Task and Group Enrichment, Data Object Enrichment*** **and** ***Predicate Enrichment:*** Automatic mapping from BPMN to BPEL is intractable without additional information. Steps 3, 5, 6 and 7 are defined to enrich the BPMN model in order to facilitate automatic generation of executable service composition from the BPMN model to BPEL. Step 5 (*Task and Group Enrichment*) includes more information about the service defined in Step 3 such as URI, business type, and so on. Step 6 (*Data Object Enrichment*) consists of refining the definition of the BPMN data objects by associating data types and assigning variables to them. Finally, Step 7

(*Predicate Enrichment*) consists of explicitly solving predicates of the BPMN model (e.g., loop, decision commands).

Steps 8, 9 and 10: ***Generic WS-BPEL Generation, Service Information Generation*** **and** ***Generic Security Specification Generation:*** In these steps, we map the annotated BPMN model into an executable composition (WS-BPEL) along with the security configurations and enforcement mechanisms. This mapping is carried out in two steps as we choose to decouple the executable composition from any particular orchestration engine. Steps 8, 9 and 10 refer to the mapping of the annotated BPMN into platform-independent (which we call *generic*) WS-BPEL, service information and security configurations, and Steps 11 and 12 yield *platform-specific* BPEL and security configurations.

Step 8 (*Generic WS-BPEL Generation*) maps the annotated BPMN into a WS-BPEL composition that is independent from any particular WS-BPEL engine. The WS-BPEL composition only contains standard WS-BPEL elements. Step 9 (*Service Information Generation*) generates a file that contains generic information (XML file) about the services to be used in the executable service composition (see Steps 3 and 5). Finally, Step 10 (*Generic Security Configuration Generation*) is responsible for yielding a generic configuration file that includes needed security configurations to realise the security requirements defined in Steps 2 and 3. Internally, NF-Actions will be actually bound to WS-BPEL commands such as `invoke`, `receive`, `reply`, `sequence`, `assign`, `variable`, `eventHandler` and `throw`. NF-Actions have (in most cases) different meanings when they are bound to different WS-BPEL elements. For example, the NF-Action *UseCryptography()* bound to the WS-BPEL command `receive` means that once the receive is the entry point of the composition, the interactions between the client and the orchestration engine must be encrypted. On the other hand, when *UseCryptography()* is bound to the WS-BPEL command `invoke`, the interactions between the orchestration engine and the service provider must be encrypted.

Steps 11 and 12: ***Platform-Specific WS-BPEL*** **and** ***Security Configuration Generation:*** In these steps, the generic files generated in Steps 8, 9 and 10 are transformed into platform-specific WS-BPEL (Step 11) and security configuration files (Step 12). Step 11 (*Platform-Specific WS-BPEL Generation*) produces a WS-BPEL composition that is customised to run on a particular orchestration engine (e.g., Apache ODE in our work). Step 12 (*Platform-Specific Security Configuration Generation*) generates a set of configuration files to enforce the security requirements in a particular orchestration engine or security module (e.g., Apache ODE Rampart in our work).

Step 13: ***Execution of WS-BPEL:*** This step runs the WS-BPEL executable composition and enforces the security requirements at specified enforcement points using an auxiliary engine, which is described in more detail in Section 5.2.

5 Architecture, Implementation and Example

This section presents the Sec-MoSC solution architecture that supports security abstractions (Section 3) and the Sec-MoSC methodology (Section 4). Following this

architecture, we have implemented a toolset to demonstrate the viability of the approach. We also detail how the prototype toolset can be used to support the modelling and development of the VTA use case introduced in Section 2.

5.1 Architecture

Figure 2 shows the architectural overview of the proposed solution, which includes a security extension of a BPMN editor (*Sec-MoSC Tool Editor*), a set of repositories (for *Security NF-Actions*, *Services* and *Log*), execution environment tools (*Auxiliary Engine* and *Orchestration engine)* and XML files.

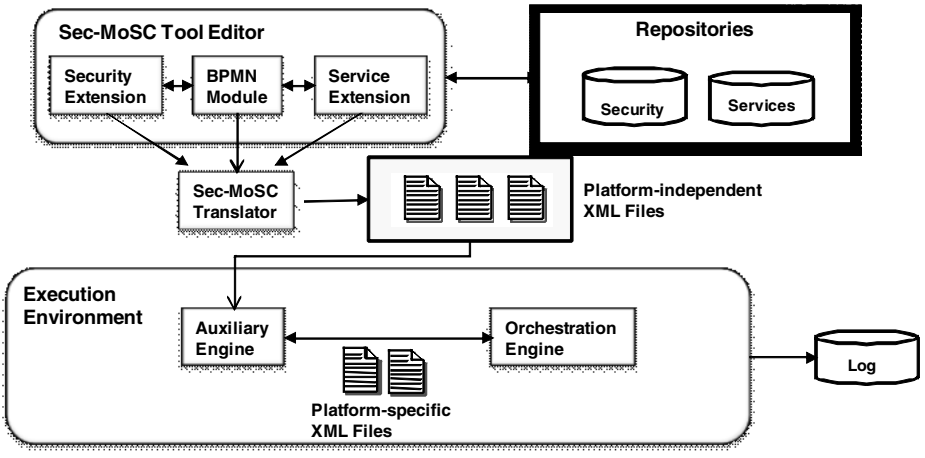

Fig. 2. Sec-MoSC Architecture

Sec-MoSC Tool Editor provides support at development time and is composed of four main components: *BPMN module*, *Security extension* module, *Service extension* module and the *Sec-MoSC Translator*. The *BPMN module* allows the developer to define a business process using the standard BPMN notation. The *Security extension* provides support to model the security requirements and bind them to BPMN elements. The *Service extension* is responsible for the annotation of service information (e.g., candidate services to execute a task) into the BPMN. Finally, the *Sec-MoSC Translator* is in charge of carrying out the mapping of the annotated BPMN model into WS-BPEL and other specification files. The repositories *Services* and *Security* store information about pre-registered services and supported security NF-Actions. These repositories support annotation of the business process model. The *Log* repository stores security log information generated at runtime.

The *Execution Environment* is responsible for executing the service composition and realising non-functional requirements. This realisation is performed by the enforcement mechanisms through generation of configuration files to the *Orchestration Engine*. This generation is performed by the *Auxiliary Engine*, which applies and manages configurations in order to execute the secure service composition. The

Orchestration Engine has the traditional role of executing the composition. The auxiliary engine has an internal component, the *MoSC Security Module* , that executes the security enforcement mechanisms by implementing needed NF-Actions either directly or by configuring existing engines such as Apache Rampart [1]. We provide three complementary views in the Sec-MoSC Tool Editor corresponding to the three levels of abstraction: business, design and execution. In the business view, the business user can use the editor to annotate a BPMN model with provided security annotations. The design view shows the corresponding BPEL code and security configuration files to the annotated BPMN model. The developer can inspect the generated code and further refine the configuration of NF-Actions. The execution view allows the user to deploy the composition logic into the Execution Environment and monitor and inspect the log of secured composition execution.

It is important to note that the functional BPEL code is not altered in order to insert enforcement points. In the presented approach, the enforcement points are specified in the configuration files generated by the Auxiliary Engine. For example, if the user specifies that a specific communication should be encrypted via the NF-Action of *UseCryptography*, the Auxiliary Engine will receive this requirement and will generate specific configurations in order to guarantee that the enforcement points will be performed. In this example (*UseCryptography*), the execution of the NF-Action will alter the SOAP request and SOAP response messages by encrypting/decrypting the message. This process is transparent to the functional execution of the application. An example of a configuration file will be presented in Section 5.3.

5.2 Implementation

Components of the Sec-MoSC architecture presented in Figure 2 are implemented as an Eclipse plugin. The *BPMN Module* was implemented by extending the BPMN editor [7] to support the BPMN concept of *TaskType* (this concept is present in the BPMN 1.2 specification). We used Eclipse GEF framework to implement the definition and binding of security and service information to the BPMN model. These extensions access the *Service* and *NF-Action* repositories at development time to assist the model annotation. The *Sec-MoSC Translator* has been implemented considering the BPMN elements and user annotations in the Sec-MoSC Tool Editor in order to generate platform-independent files containing all information required for the execution of the service composition and configuration of security mechanisms.

Figure 3 shows a screenshot of the *Sec-MoSC Tool Editor*. This figure is divided into three main areas. The palette on the right includes BPMN shapes (Task, Predicate and Data Object) and security shapes (NF-Attributes represented as cloud, NF-Statements as locker, Group of NF-Actions as dashed rectangle and NF-Actions as solid rectangle) used in the graphical modelling. The modelling area in the centre contains the business process and security definitions. The property area shows the properties of a particular BPMN shape (e.g., size) or security shape (e.g., *Encryption Type* for this particular example shown in the figure). The property area also includes service information bound to a BPMN task.

The *Auxiliary Engine* has been implemented in Java and it is able to generate both a *Platform-Specific WS-BPEL* (from the *Generic WS-BPEL* and service annotations) and a *Platform-Specific Security Configuration* (from *Generic Security Specification*);

and intermediate invocations/responses to/from the *Orchestration Engine*. The *Orchestration Engine* used is the Apache ODE [2] (based on Axis 2) due to its wide adoption and its support for security enforcement points provided by the Rampart security module [1]. Finally, the *Auxiliary Engine MoSC Security Module* was implemented based on Axis 2 that implements the NF-Actions not supported by Rampart, e.g., *RestrictAccess* and *Log* (see Table 1).

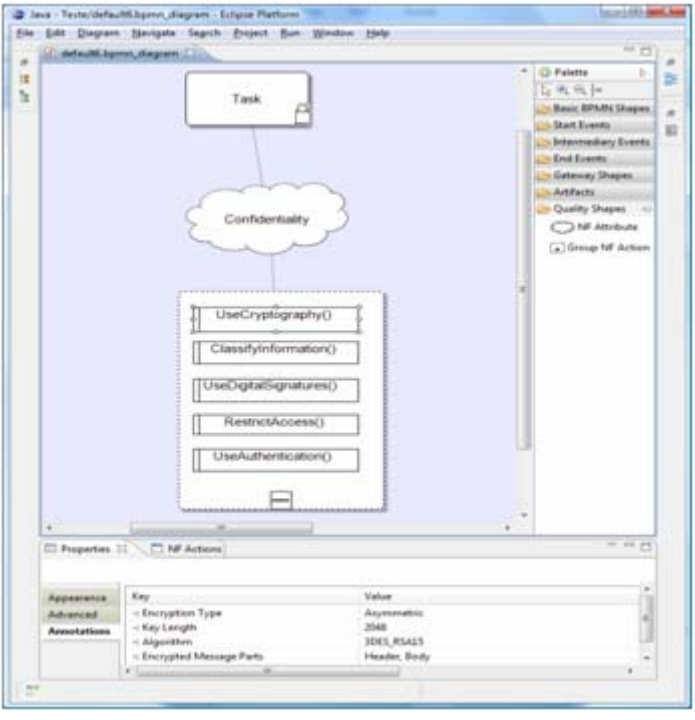

Fig. 3. Sec-MoSC Tool Editor

Figure 4 shows the architecture of *Execution Environment*. This architecture uses the orchestration engine Apache ODE [2], built under the framework Axis2 [17]. The service composition principle is based on the Axis2 message flow. The functional part of the application is translated to WS-BPEL 2.0 and is executed by the Apache ODE engine. The non-functional part of the application is handled by interceptors associated to the NF-Actions. When the Apache ODE sends/receives an invocation, the SOAP message crosses the Axis2 message flow; if the service to be invoked has an associated NF-Action, a handler will be invoked in order to perform this NF-Action. The Apache Rampart is able to handle a specific set of NF-Actions, e.g. *UseCryptography*, *UseAuthentication* and *CheckDataIntegrity*. However, there are some NF-Actions that Rampart is unable to manage; in this case, the Auxiliary Engine takes

over the responsibility. The NF-Actions *RestrictAccess* and *Log*, for example, are managed by the Auxiliary Engine interceptors.

Note that the flow of messages inside the *Execution Environment* depends on the security requirements defined. For example, the implementation of the NF-Action *RestrictAccess* receives an invocation and then forwards it to the Auxiliary Engine that checks if the IP address in the output message has some kind of constraint. If the constraint is not satisfied, the message is not forwarded through other elements in the FlowOut. Otherwise (the IP address is allowed), the message is forwarded to Rampart that may enforce other NF-Actions (e.g., *UseCryptography*). Next, Rampart forwards the message to the web service that actually handles the request.

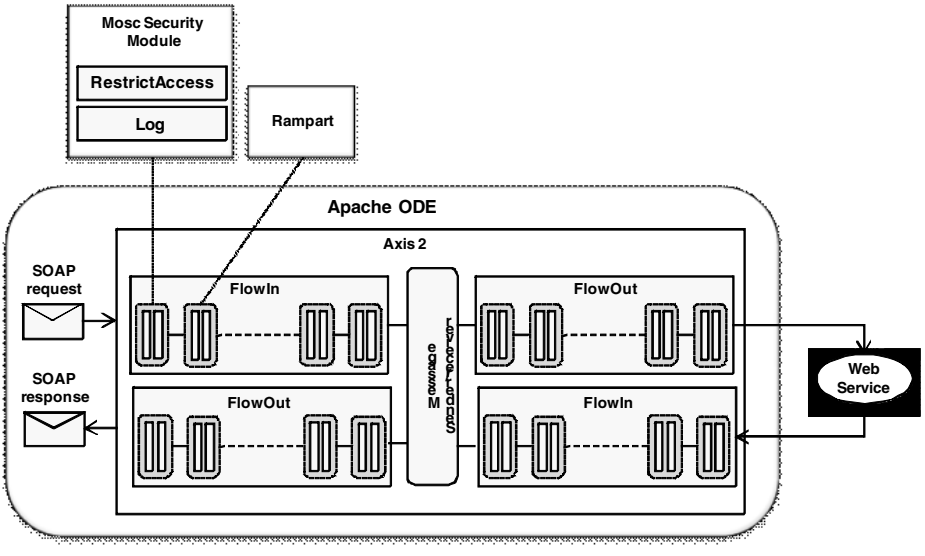

Fig. 4. Implementation of the Execution Environment in the Apache ODE

5.3 Illustrative Example

This section presents how the Sec-MoSC methodology and the tools have been used to model the example scenario presented in Section 2. We describe a use case in which the user buys a national air ticket. The security requirements defined in this business process are *Confidentiality* (NFR 01) and *RestrictAccess* (NFR 05), which are implemented by the NF-Actions *UseCryptography* and *RestrictAccess*.

Figure 5 presents the resulting business process model after steps *Business Process Modelling* and *Security Requirements Modelling*. This business process includes four Service Tasks (*Receive customer data*, *Check flight availability*, *Request authorisation payment* and *Send confirmation e-mail*), the NF-Statements bound to *Receive customer data* (locker) and *Request authorisation payment* (locker), the NF-Attribute *Confidentiality* and the NF-Actions *RestrictAccess* and *UseCryptography* that implement this NF-Attribute.

In step *NF-Action Enrichment* (shown in Figure 1), the properties of *UseCryptography* and *RestrictAccess* are configured. The next enrichment steps (*Data Object Enrichment*, *Task and Group Enrichment* and *Predicate Enrichment*) collect additional information from the users such that the remainder of the process can be automated. To perform the generic generation steps 8, 9, and 10, the *Sec-MoSC translator* takes the annotated BPMN model as input and generates three platform-independent files, which we term generic files. The security information file includes information for specific security enforcement actions expressed in the model. The service orchestration specification file is a platform-independent version of the functional executable code (WS-BPEL code). The service information file includes information (e.g. URI, partner links, operation names) needed to enrich the service orchestration specification file to perform calls to the selected services.

The steps *Platform-Specific WS-BPEL* and *Platform-Specific Security Configuration Generation* generate four files: one *Platform-Specific WS-BPEL* (executable service composition) and three security configuration files. The security mechanisms to support the NF-Action *UseCryptography* are already implemented by Rampart, which means that for this particular NF-Action we generate a Rampart configuration file that enforces this NF-Action. The NF-Action *RestrictAccess* is not supported by Rampart. Thus we implement it using an *Axis2 phase*.

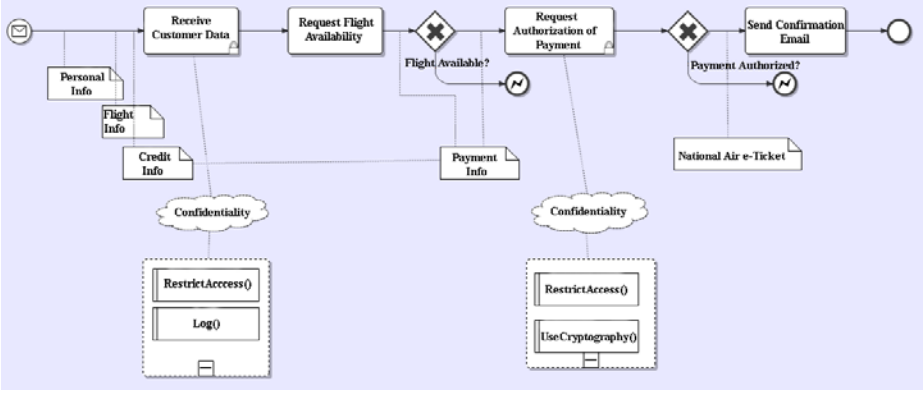

Fig. 5. Business Process and its security requirements

The following XML code refers to the specification of the security mechanism related to the NF-Actions *UseCryptography* and *RestrictAccess*. This code is generated by the Auxiliary Engine and provides configurations to the Apache Rampart (Line 9 to Line 11, and Line 17 to Line 19), in the case that the NF-Action can be implemented using this security module, and to the MoSC Security Module, in the case that Rampart do not provide support to the NF-Action (Line 21 to Line 27).

With respect to the NF-Action *UseCryptography*, this file specifies that the encryption algorithm TripleDESRSA15 (line 10) must be used. In addition, at line 18, it is stated that the body of the message must be encrypted. All of this information is used by Apache Rampart to guarantee that the user requirements will be respected.

In the case of the NF-Action *RestrictAccess*, the Apache Rampart does not support the enforcement. Therefore, the MoSC Security Module, part of the Auxiliary Engine, is responsible for the enforcement of this NF-Action. The associated configuration specification to this NF-Action can be found from Line 21 to Line 27 of the following code.

```
(1)  <wsp:Policy wsu:Id="SigEncr"
(2)  ...
(3)  <wsp:ExactlyOne>
(4)    <wsp:All>
(5)            <sp:AsymmetricBinding>
(6)                   <wsp:Policy>
(7)                   ...
(8)                   <sp:AlgorithmSuite>
(9)                                <wsp:Policy>
(10)                                   <sp:TripleDesRsa15 />
(11)                               </wsp:Policy>
(12)                  </sp:AlgorithmSuite>
(13)                  ...
(14)                  </wsp:Policy>
(15)           </sp:AsymmetricBinding>
(16)           ...
(17)           <sp:EncryptedParts>
(18)                  <sp:Body />
(19)           </sp:EncryptedParts>
(20)           ...
(21)           <secmosc:SecMoscConfig>
(22)              <secmosc:restrictAccess>
(23)                  <secmosc:restrictPartner policyType="allow">
(24)                      <secmosc:destinationEndpoint
    address="172.17.0.0/16"/>
(25)                  </secmosc:restrictPartner>
(26)              </secmosc:restrictAccess>
(27)           </secmosc:SecMoscConfig>
(28)   </wsp:All>
(29) </wsp:ExactlyOne>
(30) </wsp:Policy>
```

The MoSC Security Module is also responsible for the enforcement of the NF-Action Restrict Access. When a service invocation is realized by the orchestration engine, the request message is intercepted by the MoSC Security Module that will call the handler to perform the NF-Action. The handler verifies whether the IP address of the web service to be invoked is valid based on the defined configuration (Line 24).

We learned that this model-driven approach facilitates the communication between the business analyst, security expert, and service developer to specify and refine security requirements. Having an intermediate platform-independent specification helps in both code generation and enforcement. In particular this enables the Auxiliary Engine to manage different orchestration engines and security modules.

6 Related Work

Existing research related to service composition and security usually concentrates on supporting security at one of the abstraction levels: the business process definition [10][14][11][12], the executable composition [3][18][8][5][6], or enforcement of specific security requirements at execution time [10][16].

At the business process modelling level, Menzel [10] and Rodriguez [14] have extended BPMN modelling notations with specific security elements to incorporate security requirements into business process specification. In contrast our modelling notation is minimal and generic (NF-Attribute, NF-Action and NF-Statement) and can be used to express any security requirement. Therefore, it is easily extendable. Basin [3] has complementary work focused on software modelling which offers specific notations for modelling access control requirements. Our approach handles all types of security requirements for business processes. In Neubauer's approach [11][12] the treatment of security requirements starts before defining the business process by supporting decision makers with elicitation of security requirements. Carminati [3] proposes an ontology-based method for modelling security requirements and provides a security vocabulary. In another approach, Phan et al [20] provide a framework for specification of high level security requirements as policy objectives and then translating them into WS-Policy fragments. However, they do not consider business processes and composite services, but focus on policies for atomic services.

Although conceptually a business process is mapped into an executable process, in most existing work that supports incorporating security requirements at execution time, the business process is directly defined using an executable language like WS-BPEL. For example, [18][8][5][6] extend WS-BPEL with security capabilities. Song [16] adopts an aspect-oriented approach that interprets the WS-BPEL process and plugs in calls to the security web service component. Menzel [10] reports generating only WS-SecurityPolicy fragments. In contrast, we automatically generate platform-independent configurations which are then automatically translated into platform-specific configurations which are enforced at execution time by security modules.

Another key advantage of our approach is that the extension to BPMN notation is managed separately from the BPMN model. This allows standard tools to be used with the BPMN model. Similarly, we do not modify the BPEL code corresponding to the BPMN model but rather identify the enforcement points and mechanisms in separate configuration files.

7 Conclusions and Future Work

In this paper, we presented a novel holistic model-driven approach and a toolset that supports security requirement modelling, automatic code generation and enforcement of security requirements. It facilitates the jobs of business users, security experts and developers of service composition solutions. The main contributions of the proposed approach include the definition of a set of non-functional abstractions to express security requirements at different abstraction levels, a methodology to incorporate security requirements and service information into BPMN, mappings of security and service information into executable elements, and providing execution support for security

requirements. We have prototyped the Sec-MoSC solution architecture, which includes a security-extended BPMN editor to support service composition at modelling time, and an auxiliary engine that coordinates with security enhancement modules on top of Axis 2 to realise the security requirements at execution time.

In terms of future work, we concentrate on three areas: (1) to extend the binding of security requirements to other BPMN element types beyond tasks, task groups, and data objects; (2) to define and realise additional security requirements; and (3) to monitor security requirements at runtime in such a way that when a particular requirement is violated, the execution environment can provide evidence of and possibly correct violations.

Acknowledgement. This research is supported by Hewlett-Packard Brasil Ltda. using incentives of Brazilian Informatics Law (Law nº 8.2.48 of 1991).

References

[1] Apache Software Foundation (2008), Apache Rampart – Axis2 Security Model, http://ws.apache.org/rampart/ (last visit at May 3, 2009)

[2] Apache Software Foundation. Apache Orchestration Director Engine (ODE), http://ode.apache.org/ (last visit at May 3, 2009)

[3] Basin, D., et al.: Model driven security: From UML models to access control infrastructures, ACM Trans. Software Eng. Methodology 15(1), 39–91 (2006)

[4] Carminati, B., Ferrari, E., Hung, P.C.K.: Security Conscious Web Service Composition. In: Proc. International Conference on Web Services ICWS 2006, pp. 489–496 (2006)

[5] Charfi, A., Mezini, M.: Using aspects for security engineering of Web service compositions. In: Proc. IEEE International Conference on Web Services ICWS 2005, pp. 59–66 (2005)

[6] Chollet, S., Lalanda, P.: Security Specification at Process Level. In: Proc. IEEE International Conference on Services Computing (SCC 2008), pp. 165–172 (2008)

[7] Eclipse Foundation (2008), The BPMN Modeler, http://www.eclipse.org/bpmn

[8] Garcia, D.Z.G., Felgar de Toledo, M.B.: Ontology-Based Security Policies for Supporting the Management of Web Service Business Processes. In: Proc. IEEE International Conference on Semantic Computing, pp. 331–338 (2008)

[9] Han, J., Kowalczyk, R., Khan, K.M.: Security-Oriented Service Composition and Evolution. In: Proc. 13th Asia Pacific Software Engineering Conference APSEC 2006 (2006)

[10] Menzel, M., Homas, I., Meinel, C.: Security Requirements Specification in Service-Oriented Business Process Management. In: Proc. ARES 2009 (2009)

[11] Neubauer, T., Heurix, J.: Defining Secure Business Processes with Respect to Multiple Objectives. In: Proc. ARES 2008, pp. 187–194 (2008)

[12] Neubauer, T., Heurix, J.: Objective Types for the Valuation of Secure Business Processes. In: Proc. Seventh IEEE/ACIS International Conference on Computer and Information Science ICIS 2008, pp. 231–236 (2008)

[13] Ouyang, C., et al.: Translating BPMN to BPEL (2006), http://code.google.com/p/bpmn2bpel/ (last visit: May 10, 2009)

[14] Rodriguez, A., Fernández-Medina, E., Piattini, M.: A BPMN Extension for the Modeling of Security Requirements in Business Processes. IEICE - Trans. Inf. Syst. E90-D(4), 745–752 (2007)

[15] Rosa, N.S.: NFi: An Architecture-based Approach for Treating Non-Functional Properties of Dynamic Distributed Systems, PhD thesis, Centre of Informatics, Federal University of Pernambuco (2001)
[16] Song, H., Sun, Y., Sun, Y., Yin, Y.: Dynamic Weaving of Security Aspects in Service Composition. In: Proc. Second IEEE International Workshop Service-Oriented System Engineering SOSE 2006, pp. 189–196 (2006)
[17] Tong, K.K.L.: Developing Web Services with Apache Axis2, TipTec Development (2008)
[18] Wang, X., Zhang, Y., Shi, H.: Access Control for Human Tasks in Service Oriented Architecture. In: Proc. of ICEBE 2008, pp. 455–460 (2008)
[19] White, S.A.: Introduction to BPMN, Technical report, IBM Corporation (2004)
[20] Phan, T., Han, J., Schneider, J.G., Wilson, K.: Quality-Driven Business Policy Specification and Refinement for Service-Oriented Systems. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 5–21. Springer, Heidelberg (2008)