Transparently adding security properties to service orchestration

Stéphanie Chollet, Philippe Lalanda, André Bottaro
Laboratoire Informatique de Grenoble
F-38041, Grenoble cedex 9, France
(Stephanie.Chollet, Philippe.Lalanda, Andre.Bottaro)@imag.fr

Abstract

In this paper, we present a tool allowing the design of service orchestration at a high level of abstraction. This tool also allows specifying security properties, also in an abstract way, as annotations on the services orchestration. Our purpose is to generate code, including security code, from this specification. This is especially important in the context of complex execution platforms like Enterprise Service Buses. Our tool is used in the context of industrial plants using more and more services like DPWS to implement applications.

1. Introduction

The emergence of the Internet has paved the way for the seamless integration of business and manufacturing applications. Communication medium and protocols that have become predominant in the field of Information Technology (IT) are now gaining acceptance in industrial environments. For instance, industrial Ethernet and TCP/IP-based protocols are today used in the plant floor for device networking and information exchange. Similarly, Web service technologies (like SOAP or WSDL - see www.w3.org) which are accepted as the solution of choice for application integration are now considered with interest in the manufacturing field.

The Service-Oriented Computing (SOC) paradigm promotes the use of well-defined software units - services - to support the rapid development of applications. The central objective of this approach is to reduce dependencies among composition units, where a unit is typically some remote functionality accessed by clients. By reducing such dependencies, each element can evolve separately, so the application is more flexible than monolithic applications. The visionary promise of SOC is a world-scale network of loosely coupled services that can be assembled with little effort in agile applications that may span organizations and computing platforms [8][9]. In practice, services are assembled in a service-oriented architecture (SOA) that provides mechanisms and rules to specify, publish, discover and compose available services. Two major architectural approaches have been proposed for service composition. The first approach, specifically in Web Services, offers process-oriented solutions to this issue. Other researchers have proposed service-oriented component models to build applications made of interacting components implementing services. In this approach, an application appears as a set of collaborating services that can be centralized on a single platform [5] or distributed in an Intranet or in the Internet.

Whatever the architecture is, service composition turns out to be complex for several reasons. First, there are today many technologies for describing, publishing and composing services. Different protocols and mechanisms have been defined in recent years. Let us mention for instance Web Services (www.w3c.org), UPnP, DPWS (Devices Profile for Web Services) or OSGi (www.osgi.org). These technologies require deep expertise. Cross-technology applications require almost unavailable skills. In addition, a composition of services has also to reach a set of non functional requirements related to security or availability for instance. Reaching such non functional properties requires the production of complex code demanding very specific expertise. This code is often distributed between the client sides and the server sides. It is done by hand and, as a consequence, highly error-prone and very difficult to maintain. This is clearly a brake to the goal of service reusability.

Security is one of the most important non functional aspects to be tackled when composing services to be executed in the plant floor. Security is a major concern for several reasons. First, data produced by business or control applications are generally private and have to be protected. The second issue is even more serious: since control applications can act upon the environment, malicious use of devices can have dramatic consequences. Security is then a non negotiable feature for most customers (application owners) and represents clearly an important brake to the use of service-oriented applications in the plant floor.

Innovative middleware have been recently proposed to

ease the development of distributed service-oriented applications. In particular, Enterprise Service Buses (ESB) can be seen as a new class of integration tools essentially relying on the Web services technology and associated standards. An ESB is a software bus for services which facilitates connectivity management. It allows services to connect to a single point, the bus, which is in charge of the processing of the communications. Interactions are realized through an asynchronous message oriented middleware. ESBs allow the definition of mediation operations, including transformation, dynamic routing or security functions. Note that the main standard in the domain today is Sun Java Business Integration (JBI)[1].

Regarding security operations, commercial ESB mostly provide proprietary solutions (BEA ALSB or IBM for instance). Emerging open source products such as Apache ServiceMix[2], ObjectWeb Petals[3] or Codehaus Mule[4] tend to provide more open solutions based on JBI. But, in all cases, the security solutions are very technical and technology-driven. Security chains are hard to build, deploy and maintain. They are also uneasy to change and to reuse. We believe that there is a clear need to separate the definition of security operations from ESB mechanisms. In the remainder of this paper, we present an approach based on models and compatible with the ESB philosophy.

In this paper, we present an original approach to deal with security in service-oriented applications. It is based on the use of a tool allowing the orchestration of abstract services and on the separate definition of security constraints. Code generation is used to obtain a secure orchestration of concrete (actual) services. This research is conducted within the SODA[5] project. The paper is organized as follows. First, our approach is presented. Then, proposition for the approach is detailed. Section 4 presents an example illustrating our approach. Section 5 presents related work. Finally, section 6 concludes this paper.

2. Approach

2.1. Principles

A generative approach have been implemented where services orchestration is expressed in an abstract way and where security properties are clearly separated from the orchestration specification. Actual orchestrations are generated from these two specifications (process and security) and appropriate code is deployed on the runtime environments (an ESB for instance).

[1] http://www.jcp.org/en/jsr/detail?id=208

[2] http://servicemix.org/site/

[3] http://wiki.petals.objectweb.org/

[4] http://mule.mulesource.org/

[5] SODA is a ITEA European project partially funded by the French Ministry of Industry.

To that end, we have clearly separated two specification tasks: the specification of the services orchestration and the specification of security constraints on the orchestration. These tasks are performed by potentially different people and are characterized by different requirements:

- A service orchestration is specified by a domain expert, an architect, who clearly knows what activities have to be performed in a given context. The domain expert needs to specify the activities in a rather abstract way since he/she has usually little knowledge about the activities implementations. A clear separation has to be maintained between activities and their implantations that can change overtime.

- Security constraints are expressed by a domain expert, a security architect, who clearly knows the security properties that have to be met at the different stages of a service orchestration. This domain expert, that can be a different person than the one mentioned in the previous item, is not an expert in security technology. He/She therefore needs to express the security properties in abstract terms. Indeed, domain experts know the security constraints that must be attached to different services, like a DPWS device for instance, but they are usually unaware of the precise technologies that have to be used (encryption algorithms, certificates, tokens, etc).

2.2. Abstraction

As previously said, we believe that service composition should be expressed in an abstract way in order to allow designers to focus on the important aspects (as opposed to noisy technical details). To do so, we have decided to use the APEL tool [6] that allows the orchestration of activities that are abstract description of an action to be performed. More precisely, APEL defines three main concepts explained in Figure 1:

- An activity is a step in the process that results in the execution of an action. An action can be of any type including the call to a service (a Web service or a DPWS service for instance),

- A product is the element (data type) which flows between activities,

- A resource plays one or more role in an activity.

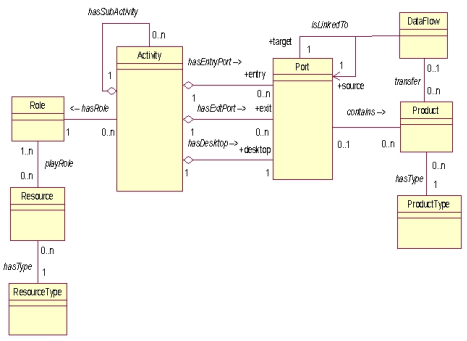

Figure 1. APEL meta-model.

Relationships between these concepts are specified in a meta-model. APEL clearly separates the different concepts that are manipulated by an orchestrator, or a process engine. It also defines concepts at a level of abstraction that allows to specify orchestration in abstract terms rather than in very specific, technology-tainted way (like most BPEL engines for instance). Finally, as explained hereafter, it allows us to express security on the most relevant items

2.3. Security properties

As previously introduced, a security architect expresses security constraints directly on the service orchestration and at a high level of abstraction. We have currently restricted security properties to the following ones:

- Authentication and authorization properties permit to guarantee the identity of person (the user) and the origin of information,
- Data integrity property allows to guarantee that some data has not been changed, destroyed, or lost in an unauthorized or accidental manner,
- Confidentiality property guarantees that some information is not made available or disclosed to any unauthorized parties.

Security design needs a knowledge of threats and risks generated by the loose coupling of services. Security architect defines the policy required to protect resources from threats in function of accepted risks. Policies are numerous and countermeasures that enforce policies are combination of security items.

3. Proposition

3.1. Separate views

The APEL orchestrator have been extended in order to allow the specification of security properties on the service orchestration in terms of annotations. Hence, two distinct roles can be distinguished when developing a composition. A domain expert (an architect) defines the service workflow. From this specification, a security expert (in the domain) can add expected security properties.

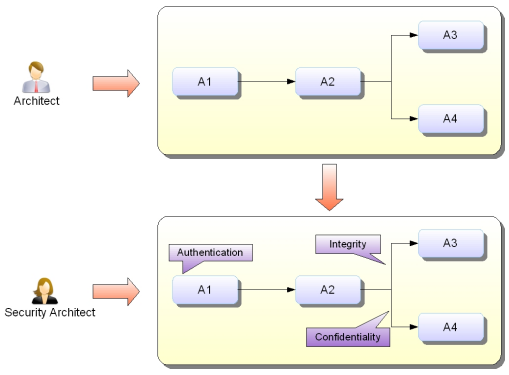

Figure 2. Orchestration designers.

Figure 2 shows how and where the different actors intervene. At first, domain designers (architects) provide an orchestration (top of the figure). Then, domain experts in security add annotations to the orchestration in order to provide mandatory security features. In this example, authentication is required for activity A1. The product from activity A2 to activity A4 must be confidential. The integrity of product from activity A2 to activity A3 is required.

Security properties are expressed on the orchestration at a high level of abstraction. For instance, an activity needs ”authentification”. No information regarding the way security constraints are implemented is provided. It is up to a code generator to provide the necessary glue code (or mediation code) allowing to deal with security constraints when running an orchestration.

The code generator is a difficult part of our approach since it may require complex code to be written, especially when running environments are distributed. However, once done for a given environment, the code can be considered as correct, safe and, eventually, optimal.

3.2. Annotation validity

A major concern when annotating a process with non-functional properties is the validity of these annotations. We believe that, in our case, it is very important to verify that the constraints make sense in terms of security in order to assist the designer and also to ensure a valid code generation.

The approach to do so is to:

1. create tools directed by models, which means that the tools we use can be seen as model editors like in the MDA approach,

2. express rules at the meta-model level.

This approach is illustrated by Figure 3. A meta-model of the APEL orchestrator and a meta-model of security in the domain of web services composition have been developed. The APEL meta-model defines the concepts of activity and product. The security meta-model defines the notions of authentication, integrity and confidentiality.

These meta-models are taken as inputs by design tool. This ensures that the orchestration and the security annotations that are created by a designer are valid in domain.

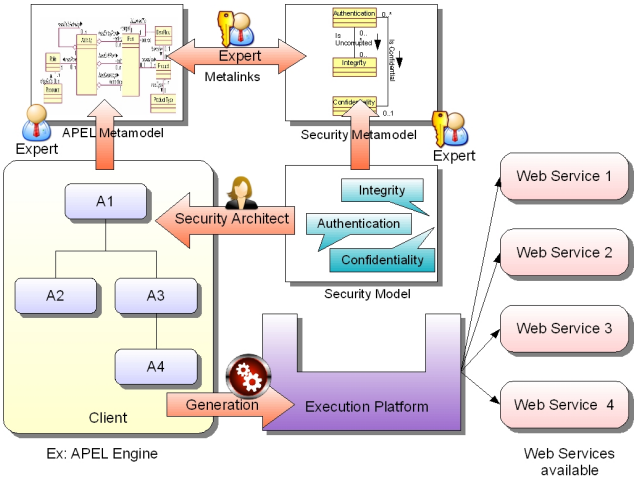

Figure 3. Meta-models approach.

The authorized relationships between concepts of both meta-models have been also defined. Thus, security annotations are always controlled by the tool. They have to be compatible with the rules expressed at the meta-models level.

Figure 4 shows the relationships that have been defined. The meta-links specify which security properties can be specified on which orchestration concepts. For instance, it appears that "authentication" can be applied only on Activity. Integrity and confidentiality are applied on Product.

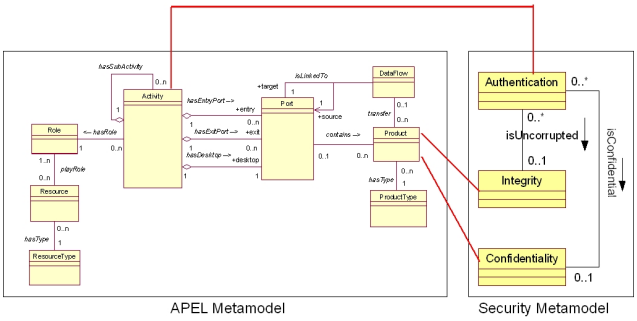

Figure 4. Meta-models links.

3.3. Code generation

The last step in approach is to generate executable code from the annotated orchestration. As illustrated by Figure 5, the execution can be made on various execution platforms. It is worthwhile noting that using an ESB as the execution platform raises additional security issues since the bus has to be protected itself. We have not been working on these issues so far.

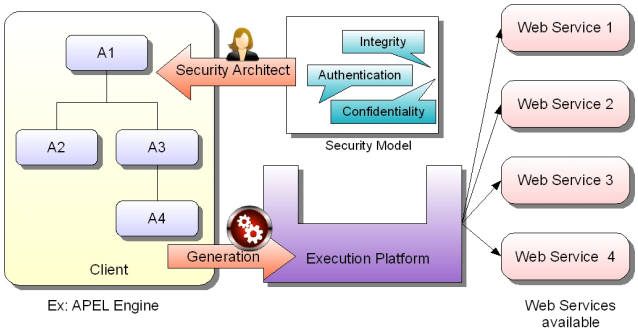

Figure 5. Code generation.

Currently, our tool generates code at the orchestration level, that is on the same machine as the one running the orchestration engine. More precisely, the generated code includes the weaving of the functional code with the security code. Aspect oriented programming is used to generate a security descriptor file compatible with WSS4J and the actual calls to web services on the client side.

4. Example

4.1. Simple scenario

Let us take a simple example in order to illustrate the approach. This example has been worked out with our industrial partner, Thales, within the SODA project.

The example takes place in a factory where we have four activities implemented by applications or specific devices in the form of Web services and DPWS services. The activities are the following:

- Acquisition: the purpose of this activity is to collect data from sensors like temperature, humidity,... It aggregates and sends data to the analysis activity.
- Analysis: this activity analyzes aggregated data and sends its results to the following activities.
- Storage: this activity records data in a data warehouse for example.

- Processing: this activity provides the appropriate reaction in response to its inputs. For example it can stop a machine.

Figure 6 shows how the four activities are orchestrated (snapshot of our tool).

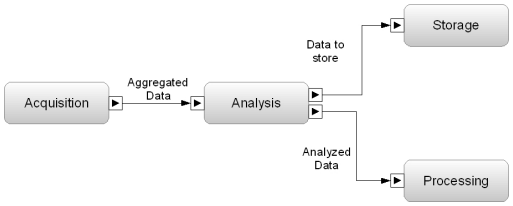

Figure 6. Business process.

Clearly, the activities presented here before have to be secured.

4.2. Adding security properties

In the previous scenario it is possible to add security properties. For example, like presented in Figure 7, we have annotated the orchestration with authentication, integrity and confidentiality properties.

At this level we do not specify how the security properties are implemented. The authentication can be ensured by a username with password or by using a X.509 certificate. As expressed on the meta-models relationships, we apply only the authentication characteristics on activities. Integrity and confidentiality are applied on data. The exact method, like username/password or X.509 certificate, is automatically chosen in function of the requirements of web service and the user's capabilities.

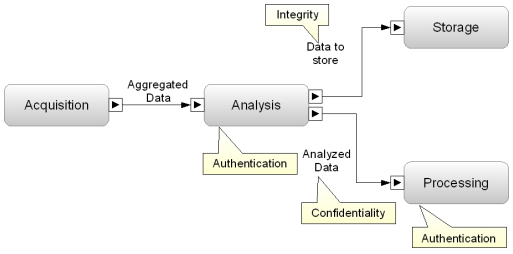

Figure 7. Secured business process.

4.3. Abstract to concrete

Once the orchestration has been specified in abstract terms, activities have to be attached to concrete services like web services or simply to java code. Figure 8 shows the mapping between abstract activities and concrete services in our example. This mapping is guided by the APEL tool: on the one hand there are abstract activities, on the other hand there are available concrete services.

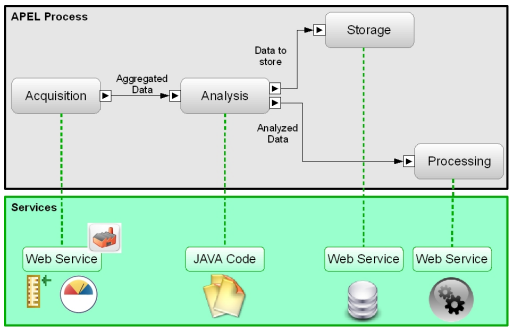

Figure 8. Mapping between abstract activities and concrete services.

Activities Acquisition, Storage and Processing are implemented as web services. Activity Analysis is implemented by Java code.

4.4. Mapping for security properties

The mapping between abstract activities and concrete services is done through attachments. Activities call services and services sends data out to activities. Services do not communicate directly. These communications are represented on Figure 9 by dotted arrows.

In concrete terms, security must be applied on the communications. More precisely, authentication is executed on service calls. Integrity and confidentiality, which are concerned with protecting data, are applied on two communications.

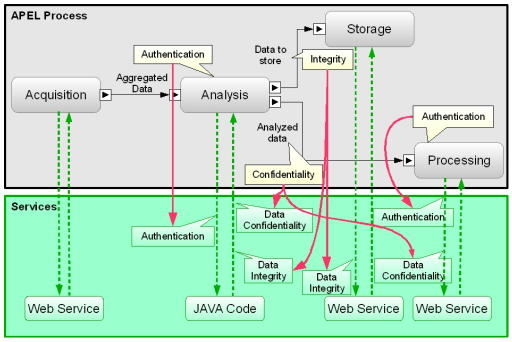

Figure 9. Security on communications.

Specific security mechanisms can be generated or chosen by the designer. In our example, we have decided to use X.509 certificate for the authentication of activity Acquisition. We have used username and password for the authentication of activity Processing. Confidentiality and integrity

can be provided by encrypted data with XML Signature and XML Encryption.

5. State of the art

Security properties for web services are provided in many specifications like Web Service Security [7], Web Service SecurityPolicy [4], Web Service Policy [2]... These standards are numerous and it is often not easy to manipulate them. However, Tatsubori et al. [10] propose a tool with a simplified, business-policy-oriented view. The tool permits to model messages between customers and business partners, lists various threats, and presents best-practice security patterns against the threats. A user can select among variations on the basic patterns according to the business policies, and then apply them to the messaging model through the GUI. The result of the pattern application is described in the Web Services Security Policy Language. This tool facilitates the adding of security properties in function of threats but it does not validate the choice of the user.

An important objective of our work is to separate the functional part of an application from the non-functional part. SCENE [3] is a service composition execution environment which supports the development and execution of service compositions able to (re)configure themselves at runtime. It extends BPEL [1] with policies and constraints for runtime platform to execute such compositions. SCENE offers a language that allows the composition to be defined in terms of two main aspects: a process part (written in BPEL), which captures the main business logic, and a part, mainly defined in terms of rules, that disciplines the way the composition evolves at runtime by applying proper binding and re-configuration policies. The decoupling between process and rules allows for a proper separation of concerns. Like in our work with APEL, the BPEL code describing the process defines the data and control flow among the various elements of the composition. This approach draws near our approach in the separation of concerns. And we can think that it is possible to create a rule file to express security characteristics

6. Conclusion

In this paper, we have presented a tool to specify service orchestration in an abstract manner and to express security constraints still in an abstract way. This tool relies on code generation to make the like between abstract specifications and actual code, including security mechanisms. We believe that such an approach is very efficient to deal with complex execution environment like ESBs for instance.

The tool also relies on meta-models. This ensures the validity of the orchestrations that are specified by designers. It also ensures the validity of the security annotations that are added to an orchestration.

We are currently working on two important topics in order to answer industrial demands regarding the tool. First, we are adding an "identity management" module in order to preserve the identity of a user during a complete orchestration execution and to automatically generate the security elements requested by the invoked web services (certificates for instance). In this paper, we consider that the main actor is the user of the workflow (or an activity that can identify itself through various technologies). Thereby an "identity management" allows multiple users of the workflow. Second, we are investigating the way to integrate other non-functional aspects, logging for instance, in the tool using the same approach that is the development a meta-model and the specification of meta-links with the APEL meta-model.

References

[1] T. Andrews et al. BPEL4WS, business process execution language for web services version 1.1, 2003. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf.

[2] S. Bajaj et al. Web services policy framework (WSPOLICY), 2006. http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy0904.pdf.

[3] M. Colombo, E. D. Nitto, and M. Mauri. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. *Service-Oriented Computing ICSOC 2006*, pages 191–202, 2006.

[4] G. Della-Libera et al. Web services security policy language (WS-SECURITYPOLICY), 2005. http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf.

[5] C. Escoffier, R. Hall, and P. Lalanda. iPOJO: an extensible service-oriented component framework. *To appear in IEEE Int. Conf. on Service oriented computing*, 2007.

[6] J. Estublier and S. Dami. APEL v3 specification. *Esprit deliverable, PERFECT project, LSR/IMAG, Grenoble, France*, December 1995.

[7] A. Nadalin et al. Web services security: SOAP message security 1.0, 2004. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.

[8] M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. *WISE*, pages 3–12, 2003.

[9] SECSE. Toward service-centric system engineering. *Int. Conf. on Service oriented computing*, 2003.

[10] M. Tatsubori, T. Imamura, and Y. Nakamura. Best-practice patterns and tool support for configuring secure web services messaging. *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, 2004.