Security Meta-Services Orchestration Architecture

Aymen Baouab, Olivier Perrin, Nicolas Biri, Claude Godart

Security Meta-Services Orchestration Architecture

Aymen Baouab[1 2], Olivier Perrin[1], Nicolas Biri[2] and Claude Godart[1]
[1]INRIA/LORIA (ECOO Team), Nancy, France
{aymen.baouab,olivier.perrin,claude.godart}@loria.fr
[2]CRP-Gabriel Lippmann (ISC Dept), Belvaux, Luxemburg
biri@lippmann.lu

Abstract

SOA have been deployed as a mean to offer a better flexibility, to increase efficiency through reuse of services and also to improve interoperability by providing new opportunities to connect heterogeneous platforms. However, those benefits make security more difficult to control. Fortunately, new standards are proposed to treat this issue, but their current use makes the architecture much more complex and challenges the characteristics of SOA. In this paper, we address this issue by separating security services from business ones and organizing the architecture referring to the principle of separation of concerns. Next, we propose a new model which consists of three components: business services, security meta-services and an orchestration service. Then, we show that the architecture remains secure while enforcing its flexibility and agility.

1 Introduction

During the last few years, the architecture of IT environments has significantly changed. Legacy solutions based on huge monolithic applications lack in agility when responding to the evolution of IT and business needs. As a consequence, many enterprises have moved to service oriented architectures (SOA) aiming to achieve a more flexible system landscape, which facilitates integration of new components.

Indeed, SOA offers companies new ways to exchange data with their customers, partners and suppliers. To secure these exchanges, the use of new suitable security mechanisms is needed. In this type of architecture, third-party services are often deployed in public areas (usually non-secure) in order to simplify the exchange and promote interoperability. In this case, security at the transport level is not enough and it is therefore necessary to move up to security at the application level. To deal with this requirement, several standards (WS-Security, SAML, XACML, WS-Trust, WS-Federation, WS-Policy) which cover security issues of Web services architectures have been developed. However, their current use addresses security at deployment time and not throughout the entire application lifecycle. These specifications are difficult to integrate into each business service especially when security requirements change frequently. Implementing these specifications without defining architectural guidelines makes the architecture much more complex and challenges the main features of SOA such as agility, flexibility and interoperability. Thus, as some legacy applications exposed as business services already have their own security mechanisms, managing the overall security chain when composing these business services represents a difficult task. Security functions must be handled dynamically depending on the security requirements and the legacy security mechanisms. For instance, when considering the project of the Luxemburg National Family Benefits Funds (CNPF) in which we are involved and the technical challenges encountered while migrating its IT environment, we have to propose a new architectural model that reduces the complexity resulting from the combination of various technologies. In fact, we aim to find a better way to integrate security that provides a high degree of reusability without impacting the business layer.

Some specific approaches expose the best practices for securing SOAs. Most of them focus on the development of the Enterprise Service Bus (ESB) [2, 11] or similar middleware solutions [3]. These solutions answer the agility issues and allows for a transparent access to the services. However, each change in the IT environment should correspond to an adaptation in the organization of the ESB. The problem of agility is only moved up to the ESB, but still unsolved. To address this issue, security functions can be implemented into separated services communicating with the rest of the system by means of messages. With a such solution, security will be decoupled not only from the business logic but also from the middleware system. The problem is then to find out how to make services operating together without having unwanted consequences on the security of the whole system.

In this paper, we address these aspects by introducing a new approach that overcome these limitations concerning SOA security and we define an adequate architectural framework called ***S**ecurity **M**eta-**S**ervices **O**rchestration **A**rchitecture, **SMSOA***. This architectural framework consists of security meta-services and a central service used for the orchestration of the meta-services depending on the policy associated with the business service to be protected.

Section 2 presents the motivations. Section 3 introduces the meta-service concept. Section 4 describes the most relevant elements of the proposed architecture, their communication and their composition. Section 5 comments on some similar approaches and finally, Section 6 concludes.

2 Motivation

A key objective of SOA consists of increasing the flexibility and the agility of the business [10]. When considering security for SOA, several possibilities can be investigated. First, one possible architecture is to provide an unique security component (including authorization, authentication, audit, etc.) integrated with the ESB. This solution breaks the modular approach of SOA, and its flexibility and is difficult to adapt to the specific needs of each service. A second approach is to include security in each business service of the SOA. This solution appears to be time consuming and its cost is very high as it needs to rewrite each business service. Furthermore, with a such solution, SOA governance and security management may become much more complicated to ensure. A third approach is to provide security services that can be composed depending on the requirements of the business service to be protected. This approach is the one we have chosen. It takes its inspiration from Aspect-oriented Programming (AOP) [4]. The goal of AOP is to respond to the separation of concerns issue: many modules in a program share common components as, for instance, logging framework. These components are called crosscutting concerns, because they ”cut” across several concerns. AOP aims in finding a way to isolate these crosscutting concerns in one central place, and to facilitate their management. The idea of our approach is to apply the separation of concerns in the SOA context in order to reduce the complexity, to provide a modular approach, and to strengthen the agility of the architecture.

3 Meta-services concept

We first introduce the concept of security meta-services in SOA architectures. Meta-services do not offer business operations, but act as helper services trying to optimize and simplify business services and their design. Meta-services centralize common functions used by multiple business services. These common functions are externalized from existing business services and managed independently. Meta-services help to improve the efficiency and flexibility and also have a positive impact onto occurring costs. Concrete instances of meta-services can be for instance *orchestration services* that offer service composition functionalities, *security services* that ensure the protection of business services, or *monitoring services* that control the correct execution of other services.

In our case, the meta-services offer a security service to other services (Security as a Service). In other words, they will ensure the security functions needed for the protection of business services. By externalizing security aspects from the business logic, Web services will be loosely-connected through a messaging interface. Moreover, the meta-services will be shared by different business services. This will reduce the complexity of the architecture, increase the degree of reusability and make the security system more scalable and more portable.

3.1 Security meta-services taxonomy

The possible meta-services for handling security requirements are:

Gateway meta-service This service, located at the federation perimeter intercepts all incoming and outgoing messages and appends security relevant information.

Authentication meta-service This entity will verify that the supplied identification information is sufficient and correct. It checks the validity of authentication credentials (digital signatures, certificates, tokens, etc.).

Authorization meta-service This meta-service determines applicable authorization policy, checks message relevant information (i.e. identity attributes) and makes an authorization decision.

Audit meta-service It is designed to log certain parts of the message (passive audit) and to determine whether an exceptional state is reached. In this case, a notification describing the situation could be sent to the system administrator.

Cryptographic meta-services They are responsible for applying cryptographic functions (i.e. digital signature calculation and verification, specific parts or whole message encryption / decryption). Separating these functions from the protected business service can be justified by the fact that they require significant computational power and therefore should be implemented on more powerful machines.

Verification, validation and filtering meta-services These meta-services can be located at the federation perimeter in order to verify that the semantics of the incoming/outgoing messages is ensured. They

provide attacks prevention mechanisms (i.e. DoS, XML injection, buffer overflow, replay attacks) [2, 5]. Message filtering can be done by an XML Firewall.

Accounting meta-services These meta-services are designed to meter the usage of protected services which are not offered free-of-charge.

3.2 Security meta-services granularity

The granularity of meta-services represents an important issue that should depend on the concrete deployment scenario and the required security functions. It plays a very important role in the evolution of the agility and the performance of the whole system. A finer granularity leads to a good separation of concerns which increases flexibility and agility and makes the meta-services more reusable. On the other hand, because each intermediate service must process the message content, increasing the number of meta-services implies a higher latency due to the additional network traffic and overhead resulting from XML parsing. This results to a compromise between agility and flexibility, on the one side, and performance on the other side. Otherwise, we need to choose between a specialized meta-service and a more general meta-service more configurable and therefore less reusable.

To make the choice on the granularity, it is important to consider all the needs of the customer and the provider. This may also depend on the size of the infrastructure, the average of throughput, the distribution of services (in the same server, same network, etc.) and available resources on each machine. One must also consider the frequency of security parameters changes and the evolution of business services needs in term of security.

4 The proposed architecture

In our approach, the architecture is composed of security meta-services, an orchestrator, and business services that are located in a trusted zone. Each security meta-services implements one or more security enforcement functions, depending of the desired granularity. A message reaching the trusted zone will be intercepted by a gateway service and be routed to the orchestrator. The response message will be returned to the orchestrator to eventually invoke some security meta-services before being routed to the requester (for instance, the audit meta-service).

To ensure the security needs of a business service, we need an entity that determines which meta-services should be invoked according to the security policies. This task is done by a central point (an orchestrator) that orchestrates security meta-services depending on the destination business service. If one of the security checks failed, an exception is raised and the orchestrator stops the security process by returning an access denied to the requester. If all security functions succeed, the business service is invoked and then the response is returned to the requester. Eventually, some meta-services can be invoked after the response of the business service in order to secure the outgoing messages.

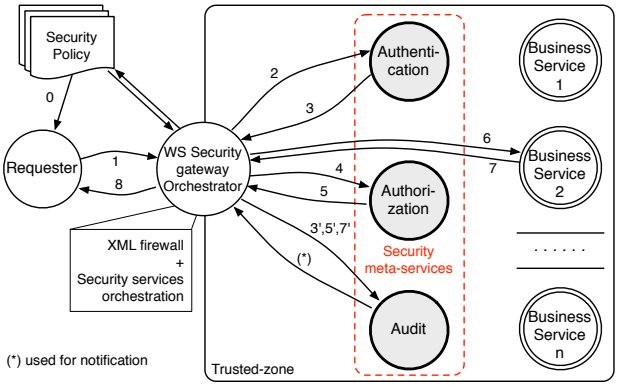

Figure 1. Meta-services orchestration

Figure 1 shows a use case scenario (steps are numbered). In this example, the requester looks for the address of the business service 2 and required security mechanisms (step 0) by invoking a public registry (i.e. UDDI). It then composes the message and invokes the service. The gateway service (here gateway and orchestrator are implemented as a single service) intercepts the message (step 1) and search for the policy corresponding to the protected service. The orchestrator invokes the needed meta-services (in this scenario, authentication, authorization and audit) and the business service respecting the security process described in the policy (steps 2,3,3',4,5,5',6,7)[1]. If one security function fails, the orchestrator stops the process and returns a deny message to the requester. Otherwise, the gateway returns the response to the customer service (step 8).

4.1 Specific orchestration

Using WS-BPEL [7], coding all combinations of possible meta-services (meta-services composition for each security policy) and updating them whenever a policy change occurs, represents a complicated task. To reduce this complexity and to improve the agility of the security system, there are two possible strategies.

4.1.1 One orchestrator per policy

This strategy introduces a specific orchestrator for each group of business services requiring exactly the same security functions. In this way, each orchestrator ensures the

[1] the prime denotes that x and x' are executed in parallel.

security of his group by invoking a static list of security meta-services whenever an incoming message reaches his area.

The gateway meta-service, located in the federation perimeter, dispatches incoming messages to the specific orchestrator according to the message destination. After receiving a message, the orchestrator invokes security meta-services and the target business service and then returns the response to the gateway service before being routed to the requester. Figure 2 shows a use case with ordered steps. In this example, zone A includes business services that require exactly three security functions while zone B includes business services that require only two security functions.

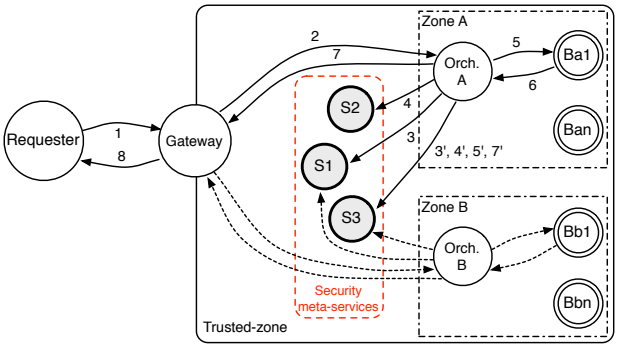

Figure 2. One orchestrator per policy

The decomposition of the federation into zones allows to distribute the load on several more specific orchestrators instead of having a central orchestrator that manages all the messages. However, there must be an orchestrator for each policy and therefore an addition (resp. deletion) of an orchestrator for each policy addition (resp. deletion).

4.1.2 One orchestrator per business service

The second strategy implements an orchestrator for each business service (Figure 3). In this way, a change in the policy might induce a change in the code of the orchestrator, but we will have a better distribution of load. Implementing an orchestrator for each business service can avoid all the problems linked to the compatibility, as we do not need to define the same interface for each business service in order to exchange messages with the orchestrator.

Note that in case of one orchestrator per policy (Section 4.1.1), all business services are connected to the same partner link and that is why they should satisfy a well-defined message format exchange and so a common interface.

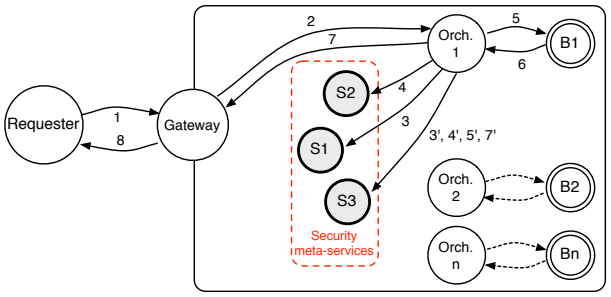

Figure 3. One orchestrator per business service.

4.2 Generic orchestration

Another strategy for orchestrating the meta-services is to use a generic orchestrator, rather than specific orchestrators as in 4.1. In this case, only one orchestrator is implemented to manage the whole system security. This orchestrator needs to be quite flexible and dynamic to permit the addition or the removal of security meta-services according to changing needs of each protected business service. Here, WS-BPEL is not well adapted because it does not allow flexibility in Partner Links. In fact, in BPEL, the partner link information is defined at design time. However, our scenarios require to define the partner link at run time, depending the policy of the business service. Oracle provides an ad-hoc solution to this problem, called *dynamic service binding*. However, the solution is conceptually poor, as all the services should be described in one WSDL file that is then used as the partner link [8]. That means for instance that no operation, message, input or output can be specified. It should be considered as a redirection.

To address this problem, we design our own orchestrator in order to be more generic (independent of the policy and the business service interface). The orchestrator should be able to carry a message according to its destination, based only on data obtained from a policy database. A change in policy and/or addition/removal of business services and security meta-services should not cause any change in the code of the orchestrator. Only a modification in the policy database will be needed.

To formally specify our architecture, we begin by distinguishing five types of components : a set of business services $\mathcal{B}$, a set of meta-services $\mathcal{S}$, an orchestrator $\mathcal{O}$, a set of policies $\mathcal{P}$ and a list of operators $\mathcal{OP}$.

4.2.1 Security process description

In order to make the orchestrator completely autonomous, we define a policy description language which will describe,

for each business service, the security meta-services to call. To do this, we will first need two operators: "//" and ",". "S_1//S_2" means that the orchestrator can invoke both services S_1 and S_2 in parallel, while "S_1,S_2" means that the orchestrator must invoke S_2 after S_1.

To simplify the security management, we group the business services according to their security functions needs. Therefore, we need two tables in our policy database: one for the correspondence between each business service and its security policy (see Figure4.A) and another for the description of each security policy indicating meta-services to invoke using our description language (see Figure4.B).

Business Service	Policy
B1	P2
B2	P3
B3	P1
B4	P2
...	

A. Table Business Service/policy

Policy	Services to invoke
P1	S1//S4,S2//S4,B//S4,S4
P2	(S1,S3,B)//S4,S4
P3	B//S4
...	

B. Table Policy/Services to invoke

Figure 4. Policy database

Figure 4 shows that for all incoming messages having the business service B_1 as destination, the orchestrator must invoke S_1, S_3 and B_1 in parallel with S_4, wait for their responses and then, if there is no exception, invokes S_4 and sends the reply to the requester.

Note that to change the policy of a business service, one can simply change its entry in the first table. To add a new business service, a new entry can be created and an existing policy can be selected. However, the second table allows to change a specific policy (i.e. add/remove of security meta-services). To add a new policy, a new entry can be created and the required security meta-services can be selected and organized. All these changes did not affect the orchestrator that will be designed to be able to parse any valid policy.

4.2.2 Policy validation

In order to avoid security process execution problems, it is necessary to validate every policy described with our language before its activation. To do this, policies should be analyzed and should satisfy a specific grammar. Using this grammar, invalid policies will be rejected by the orchestrator.

Definition 1 (Policy validation) *Let $\mathcal{S} = [S_1; S_2; ...; S_n]$, $\mathcal{B} = [B_1; B_2; ...; B_m]$ and $\mathcal{P} = [P_1; P_2; ...; P_l]$. P_i (with $0 \leq i \leq l$) is valid if it takes the following form :*

$$\{\{S_1|S_2|...|S_n\}\{//|,\}\}^* X_B \{\{//|,\}\{S_1|S_2|...|S_n\}\}^*$$

with $X_B \in \mathcal{B}$ and $P(X_B) = P_i$, and $\{A\}^*$ means that A is repeated 0 or more times.

For instance, "$S_1//S_2, B//S_3$" and "$S_1//B$" are valid policies, while "$S_1S_2, B//S_3$" and "$//S_1, B$," are not.

4.3 Security process instance state

The orchestrator needs some information (current status) to identify each received message in order to determine to which process instance the message is associated and what is the next step. Furthermore, the orchestrator may need to transmit some information (identity attributes, credentials, etc.) from one security meta-service to another. To do this, there are two possibilities. The first approach called *stateful orchestrator* is to store and manage instance identifiers and variables in the memory of the orchestrator. The second called *stateless orchestrator* appends such information to the message header (i.e. annotations).

A *stateful orchestrator* must read the destination of each incoming message, look for the security policy corresponding to this destination, extract the path described by the description language that we have defined, create an instance with a new state (in the memory) containing the necessary information and execute the process. Once implemented, the orchestrator will be autonomous as it will always be able to transform a security policy defined by our description language into a new process and be able to execute it.

With a *stateless orchestrator*, each incoming message must be annotated with a path containing an ordered list of security meta-services to invoke and an actual position indicator. This annotation represents the status of the security process and can be implemented as one or more tags of the SOAP message header. The orchestrator updates the status indicator after each invocation and then prepares himself to invoke the next meta-service. In this case, the orchestrator does not need memory to store the state of each instance. This will improve his treatment capacity and increase the throughput. In addition, with such a solution, it will have much less risks of memory crash (buffer overflow, etc.) which makes the orchestrator more reliable.

On the other hand, a *stateless orchestrator* may have to deal with some interoperability problems, especially when the orchestrator needs to call a security service located outside the trusted zone (security off-shoring). In this case, the annotation can be lost or modified by the external service which may cause disturbance in the security system. To tackle this problem, we can combine the two alternatives storing the state in the orchestrator memory only when an external service invocation occurs. This *hybrid solution* aims at combining the advantages of previous solutions to resolve problems encountered in each particular situation.

4.4 Synthesis

Since the security functions are not bound to the application and implemented into separated services, each one independent of the other, they can be shared by the different business services located in the same private network or a

trusted area. This will reduce the costs associated with implementation, facilitate upgrades and allow the small functionnal components to be better tested (i.e. using unit tests) [8]. All these advantages address the agility and the extensibility of the architecture.

Implementing the security aspect using the principle of the *separation of concerns* reduces the complexity of the whole security system and allows for a clear design easy to configure and document. Configuration may be made by the orchestration of the security meta-services according to the needs of each business service (as shown in section 4.2.1).

Another advantage of our architectural design is that the administration and the control become simpler by centralizing security. Security administrators can control the whole system by acting only on the orchestrator. This may also simplify federated authentication, federated single sign on (SSO) and federated access control.

However, performance represents a possible disadvantage to our solution. In fact, SMSOA may higher latencies because of the additional network traffic and message content processing (i.e. XML parsing by each security meta-service). However, as descibed in section 3.2, performance can be modulated by varying the granularity of services.

In conclusion, our approach has several advantages: it is extensible, it provides a high degree of reusability and it improves the agility and the flexibility of service-oriented architecures. In addition, the fact that security services are independent and can be hosted in different physical locations represents another advantage.

5 Related Work

[9] outlines a holistic approach to protecting applications and services with the use of Web Services security infrastructure. Here, some security functions are implemented into a perimeter gateway which enforce security for the entire network. In addition, a service agent will be implemented to enforce individual service security. [1] presents an approach where a single proxy acting as a gateway is implementing different security functions to enforce the security of several services. However, all these approaches does not respect the principle of separation of concerns. In [6], security functions are realized into different services and combined by means of an ESB. However, communication between security services still not designed and no architectural analysis is made.

In [8], an architectural model called SOSA was described. Security is splitted into small functional components (security services) that can be separately developed. Because components are less complex, it will be easier to reuse them. This results in a more flexible design for security system. However, combining services by means of message routing patterns using itinerary attachment decentralize the security system and make it more difficult to control and to administrate. Furthermore, it is often necessary to implement an additional service for the invocation of the protected business service (and also in case of security offshoring) which will increase latency in both directions (additional processing of the incoming and the outgoing messages). Moreover, the fact that each security service must implement a message routing mecanism represents another drawback. In fact, in addition to its task, each security service needs to determine the next service to which the message should be forwarded. This is due to the fact that SOSA is based on a decentralized message routing pattern which is called *Itinerary Routing*.

6 Conclusion

In this paper, we proposed a new architectural framework for Web services security systems named Security Meta-Services Orchestration Architecture or SMSOA. The architecture is composed of three types of components: business services, security meta-services and orchestrators. Each meta-service implements one or more security functions and is invoked by an orchestrator according to business services security policies. This results to the separation of the security aspect from business services which reduce design complexity and development costs. In addition, orchestration solution centralize the security system and therefore easier to control and administrate.

References

[1] G. Brose. Securing web services with soap security proxies. In *ICWS*, pages 231–234, 2003.

[2] D. Chappel. *Enterprise Service Bus*. 2004.

[3] Khalaf R., Nagy W., Curbera F., Duftler M.J. Colombo: Lightweight middleware for service-oriented computing. IBM Journal of Research and Development, Volume 44, Number 4, 2005.

[4] G. Kiczales. Aspect-oriented programming. *ACM Comput. Surv.*, 28(4es):154, 1996.

[5] Mark Oneill et al. *Web Services Security*. 2003.

[6] Maryann Hondo, Heather Hinton, Beth Hutchison. Security patterns within a service-oriented architecture. IBM white paper, 2005.

[7] OASIS. Web services business process execution language version 2.0. OASIS Standard, 2007.

[8] C. Opincaru and G. Gheorghe. Service oriented security architecture. In *EMISA*, pages 61–74, 2007.

[9] B. M. M. A. N. K. P. Y. Ramana Turlapati, Roger Goudarzi. Best practices for securing your soa: A holistic approach. Java Developers Journal, 2006.

[10] A. Report. State of soa adoption report – gauging the use of soa systems in the enterprise, January 2008.

[11] H. G. Woolf B. *Enterprise Integration Patterns, Designing, Building, and Deploying Messaging Solutions*. 2004.