Evolving Security Requirements in Multi-Layered Service-Oriented-Architectures

Muhammad Sabir Idrees[1], Gabriel Serme[2], Yves Roudier[1], Anderson Santana De Oliveira[2], Herve Grall[3], and Mario Südholt[3]

[1] Eurecom,
Sophia Antipolis, France
{idrees,roudier}@eurecom.fr
[2] SAP Research,
Sophia Antipolis, France
{gabriel.serme,anderson.santana.de.oliveira}@sap.com
[3] Departement Informatique,
Ecole des Mines de Nantes, France
{herve.grall,mario.sudholt}@emn.fr

Abstract. Due to today's rapidly changing corporate environments, business processes are increasingly subject to dynamic configuration and evolution. The evolution of new deployment architectures, as illustrated by the move towards mobile platforms and the Internet Of Services, and the introduction of new security regulations (imposed by national and international regulatory bodies, such as SOX[4] or BASEL[5]) are an important constraint in the design and development of business processes. In such a context, it is not sufficient to apply the corresponding adaptations only at the service orchestration or at the choreography level; there is also the need for controlling the impact of new security requirements to several architectural layers, specially in cloud computing, where the notion of Platforms as Services and Infrastructure as Services are fundamental. In this paper we survey several research questions related to security cross-domain and cross-layer security functionality in Service Oriented Architectures, from an original point of view. We provide the first insights on how a general service model empowered with aspect oriented programming capabilities can provide clean modularization to such cross-cutting security concerns.

Keywords: SOA, Evolution, AOP, REST, Security

1 Motivation and Outline

Service-oriented architectures (SOAs) constitute a major architectural style for large-scale heterogeneous infrastructures and applications that are built from loosely-coupled, well-separated services and are subject to dynamic configuration, manipulation, and evolution. Applications in service-oriented computing

[4] Sarbanes-Oxley Act of 2002 (Pub.L. 107-204, 116 Stat. 745, enacted July 30, 2002)
[5] http://bis.org/publ/bcbsca.htm

have traditionally been developed using composition in homogeneous and simple frameworks. However, service-oriented architectures do not only rely on the simple composition of services but on compositions involving multiple architectural layers, especially when the underlying platforms and infrastructures are also seen as services themselves, like in cloud computing. The rapidly increasing need to integrate business applications deployed across distinct administrative domains reflects the reality of how software is being consumed nowadays. Such applications must also be compliant with security requirements and regulations, which can change and/or evolve according to the business context. For instance, access control and monitoring for intrusion detection are prime examples of functionalities that are subject to this problem: they cannot be properly modularized, that is, defined in well-separated modules, especially if they cross administrative or technological boundaries.

Problem Statement: The problem we expose in this paper is to understand how an evolution can modify the existing service-oriented architecture with respect to the new functional and more specifically non-functional (security, trust, QoS) requirements. These requirements have to be applied consistently on the system architecture, thus involving complex service orchestrations and choreographies. They often involve invasive modifications, e.g., to enable new security functionalities that depend on and require modifications to low-level infrastructure functionalities. Hence, SOAs are also subject to evolution using *vertical composition*, that is, the coordination of multiple architectural layers over which the SOA is deployed, including operating systems, application servers, enterprise service buses, orchestration engines, etc. In contrast, *horizontal composition* consists in high level service compositions towards the achievement of business goals, typically expressed as orchestrations or choreographies.

Security analysts also need to consider threats to the underlying infrastructure and middleware for a particular SOA implementation. While it is easier to analyze the protection level at each separate layer in the SOA stack, the security properties expected from the software span across those layers. The assets to be protected originate both from the horizontal and vertical compositions. The security that may mitigate potential threats to these assets have to be deployed at different parts of that stack, and in a coordinated manner.

Aspect-Oriented Software Development (AOSD)[8,1] has emerged as the domain investigating and providing solutions for the systematic treatment of such cross-cutting functionalities. Here, we aim at exploring the combination of invasive aspects and black-box aspect compositions to secure service-oriented architectures.

The current state of the art does not propose a full fledged solution to manage, in a modular way, different security requirements affecting different architectural levels in SOAs. We propose an integrated approach, relying on AOSD, that will allow for eliciting security requirements, to formally reason about the interactions among requirements and the target service compositions, to implement security functionality as aspects, and to evolve these requirements in a controlled manner. We are building an aspect based service model, enabling the

conception of service compositions within cross administrative domains and allowing security functionalities to be modularized. In the current paper we report on the challenges we are facing in these research directions.

The remainder of this paper is organized as follows: Section 2 describes our view of multi-layered SOA architecture with related security concerns and security requirements discussed in Section 3. Section 4 presents the aspect based service model that we propose through a use case. Section 5 reviews the capabilities of the existing approaches in terms of aspects for security and adaptation of SOA in a distributed context. Finally, a conclusion summarizes the attained results and briefly presents future work in Section 6.

2 Multi-Layered Service-Oriented Architectures

Service-oriented architectures (SOAs) are considered as advanced component-based architectures for the construction of distributed systems. SOA can be seen as a continuum of different components at different levels of system abstraction, like the infrastructure, platform/middleware, and software viewpoints (cf. Figure 1). The fundamental assumption of SOA, where the service consumer needs not to worry about service implementation details and the underlying infrastructure: the availability of a software view (cf. Section 2.1 enumeration 1) to the application designer makes it possible to hide the unnecessary complexity in the implementation of business services originating from the underlying layers. Lower layers enforce different business logics, security policies, or functional constraints and have to be coordinated together with the execution of the application in order to give access to services and resources of the infrastructure. Nevertheless, as stated in [2], apart from the notion of service orientation that eases the design through loosely coupled interfaces, the service oriented architecture itself does not provide direct solutions to many of the available intricate requirements that arise while evolving such architectures.

2.1 Layering in Service-Oriented Architectures

We distinguish the following three different layers that entail three corresponding architectural views:

1. **Software View:** this layer is the most important one for the application designer. This is the place where he implements services using other services and mechanisms from lower layers. This is also the place where the application designer specifies how processes should be coordinated through the expression of orchestrations. Such a specification is generally done using a dedicated workflow languages (e.g., BPMN2.0, BPEL). In the process of orchestration the designer associates a software functionality (the services) in a non-hierarchical arrangement using a software tool (i.e., SAP Business ByDesign) that contains a complete list of all available services, their characteristics and resources, and the means to build an application utilizing

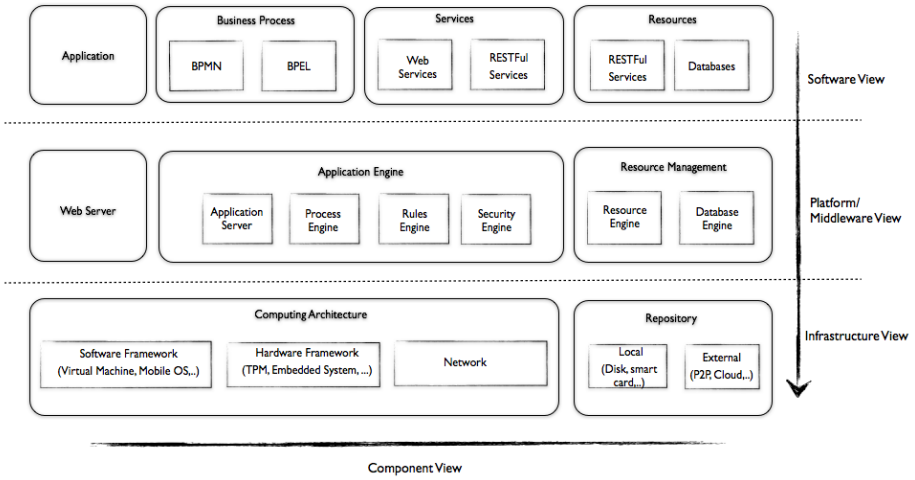

Fig. 1. A synthetic overview of Multi-layered Service-Oriented-Architecture

these sources. This specification defines how the platform layer should be configured.

Many platforms nowadays proposes Software on-demand, also known as Software as a Service (SaaS). SaaS vendors provide applications (i.e., SAP On-Demand Apps, Intalio BPM, etc.) to end users on demand. These solutions heavily rely on SOA, by abstracting every lower layer as sets of services that need to be correctly orchestrated and "consumed" by customers.

2. **Platform/Middleware View:** Service-oriented architecture middleware provides all of the facilities required to support the complete lifecycle of services. For services to operate, a middleware layer provides a collection of components (i.e., web server, application engine, or resource engine) for supporting the deployment of virtually any application. The coordination of services and the handling of the fine-grained logic of services are enforced at this layer. For instance, the definition of business rules and non functional constraints that apply to a process can be enforced by the rule engine and by the security engine components.

   In order to facilitate the deployment of business applications without considering functional and non-functional constraints (i.e., cost and complexity of buying and managing the underlying hardware and software and provisioning hosting capabilities), many platforms nowadays offer the Platform as a Service (PaaS) abstraction, like Google's AppEngine, or SAP's OnDemand Platform, for application deployment and testing. PaaS provides an additional level of abstraction emulating a virtual platform on top of the infrastructure. PaaS generally features a form of mediation to the underlying services akin to middleware in traditional communication stacks.

3. **Infrastructure View:** From the infrastructure perspective, one can distinguish between the computing architecture (software framework, hardware architecture, network, etc.) and resource repository (local or distributed) components. Any of the operations related to such low-level components must be done at this layer. For instance, the secure generation/computation/storage of cryptographic keys used to secure network communication heavily depends on the software or hardware support. The use of firewalls to control communication between two organizations is also not to be handled at the service nor platform levels.
   This point of view has been conceptualized as the Infrastructure as a Service (IaaS) approach, much touted in cloud computing, and which aims at sharing the infrastructure in order to reduce the cost of operating it. Services in that case relate to the management and customization of infrastructure mechanisms. Virtualized execution environments for the deployment of applications and distributed data storage are common examples of such services. They are being supported by an increasing range of companies (Amazon, Google, SAP, etc.) and brought to the programmer through an increasing number of service oriented APIs, notably REST ones, like for instance Amazon's services EC2 for execution environments, and S3 for distributed data storage.

The research topics we expose in this paper bring to light the need to uniformly represent, to reason about, and to deploy security (but similarly other crosscutting concerns) at the different layers of the architecture. Although these software solutions are increasingly widespread, there is no support for the modular design and evolution of the corresponding security concerns.

3 SOA Security Concerns

The preoccupation in anticipating possible security flaws in the SOAs infrastructures is fundamental for increasing the reliability of e-business applications, such that it can be widely adopted, enabling the future Internet of Services. The security analysis of SOA based business process has been extensively reviewed in the literature [14,12,11,10,16]. However, it has previously been addressed mainly from an application perspective only. In comparison, not much has been done to analyze the application level impact of attacks and vulnerabilities (also extensively analyzed but in isolation of any application) at lower layers in the SOA stack.

3.1 Attacker Model

Despite the multitude of proposed attack analysis that have hitherto been described in the literature, we take the view that existing attacker models are essentially based on vulnerabilities expressed at a single layer (more frequently low-level). We have adopted a set of practices from fundamental works in attacks on

SOA and vulnerability analysis in a new combined methodology (Multi-Layered Attacker Model - MLAM) that seems to be well-adapted to SOAs layered architectures (cf. Figure 1). The perspective we have adopted for the identification of security flaws is to have a unified approach where we consider attacks/vulnerabilities at different layers of the SOA system. The essential means that we use consist in an approach for categorizing attacks and their relationships and dependencies at different levels of abstraction, and a means for specifying a range of modalities. In order to do so, we consider a layered attack view comprising the Software attack View (SaV), the Platform attack View (PaV) and the Infrastructure attack View (IaV). Due to lack of space, we only give an abstract overview of MLAM with respect to attack(s) on multilayered service-oriented architectures. However, more detailed attack analysis and threat categorization can be found in [15,7] .

At the SaV level, security flaws are mostly related to vulnerabilities in software/application/services. These attacks rely on programming mistakes (weaknesses in the application), sometimes exploiting inadequacies between the design flaw or an implementation [14]. This is the case for injections attacks, like SQL injections, which can make it possible to read and disclose confidential data to an attacker. Similarly in WSDL scanning attacks, the access information revealed about some web sites may result in more specific and more targeted attacks.

At the PaV level, attacks rely on protocol design mistakes, on weaknesses of cryptographic primitives, on weaknesses in information processing, or sometimes on the exploitation of communication and computation implicit and harmful assumptions. Furthermore, a poor configuration and the improper use of external resources (i.e., external reference attacks) may allow an attacker to perform different denial of service attacks or information theft.

At the IaV level, an attacker might target the infrastructure by abusing resources in unforeseen ways. For instance, attacks on the power consumption, computing time, or electromagnetic emissions may lead to find out what operations are performed or what is the value of some sensitive (in particular cryptographic) data. Furthermore, the possibility of a resource compromise (injecting/altering data) significantly increases in infrastructures based on decentralization (e.g., P2P, Cloud computing), due to the architecture and to the complexity or weakness of the system with respect to operations like access controls, which may require anonymous accesses, or which may be too inflexible to configure efficiently; or authentication, in particular, when the actual deployment relies on reusable tokens/passwords, or even worse on cleartext authentication and/or transmission of content.

Merging the results of these three viewpoints: SaV, PaV and IaV helps to both identify and classify attacks on different assets as well as to highlight their relationships at different layer of the service composition.

3.2 Security Requirements

The deployment of an application requires the definition of the security properties that it should meet. This is typically performed on the basis of a security

policy, which aims at providing a high-level abstraction to the application developer. In the following we present the inherent complex implications of seemingly simple security properties in a multi-layered SOA system.

- **Integrity** applies to a quantum of information between two observations (defined, e.g., by a time and a location in the system). The property is satisfied when the quantum of information has not been modified between the two observations. This property should be monitored of course to check whether a message sent between services has not been altered, but also with respect to guarantee that the content of a storage facility has not been modified between two given read operations, or even to ensure that the execution of the software implementing a service is not being attacked through a modification to the execution environment or the code it runs.

- **Confidentiality** applies to a quantum of information and a set of authorized entities. The property is satisfied when the authorized entities are the only ones that can know the quantum of information. Thus, confidentiality properties defined at a high level should be translated at the resource level and will result in defensive security requirements with respect to data encryption, access control policies to data, and data placement.

- **Availability** applies to a service, platform, or a physical device providing a service. The property is satisfied when some service is operational. The property can be further detailed with the specification of a period during which the availability is required and of a set of client entities requesting the availability. Availability properties may impose requirements on the security mechanisms that can be implemented on a particular application, platform (like, availability of the specific process engine i.e., SAP Netweaver), or execution environment depending on the CPU, memory, networking capabilities of the considered environment or the availability of specific cryptographic functions (accelerated or not).

Other security properties that may be attached to services have a similar impact at the different levels of the SOA stack. In addition, due to their high level of abstraction, SOA applications often introduce new security properties that are themselves composing several primitive security properties: for instance, the concept of separation of duty may rely on the sequencing of two authentications.

4 Towards an Aspect-Based SOA Model

In the previous section, we introduced our concept of multi-layered SOA and briefly outlined how security concerns may have implications at different levels of a layered SOA stack. Evolution of services and notably their security requirements are hard to implement existing infrastructures, such as generic libraries. We claim that invasive modifications are needed to obtain an overall and consistent security across the system. In the following, we are discussing an Aspect-Oriented approach to ensure a systematic and consistent handling of security

concerns. We articulate this section with an integrity-related use-case that deals with certifying the validity of different components as shown in Figure 1.

4.1 Implications of the Integrity property on the SOA stack

While designing one application, an architect identifies sensitive information he wants to protect from non-authorized modifications. In the following, we assume this sensitive information is a Customer object. He wants to introduce integrity mechanisms throughout the architecture to prevent attacks whenever a Customer object is used. Introduction of integrity involves different components, that depends on several layers. Modifications to enable such property are invasive and affect several modules.

To achieve the overall integrity, we envision modifications at the three views represented in Figure 1. Software view is impacted to ensure correct handling of data when the Customer object is part of a collaboration. For example, if a business process has a step that transmits Customer data over the network through a Web Service invocation. Protecting integrity at this stage can be realized with WS-Security standards that provides message integrity. Albeit the application layer is protected against alteration over communication, the Customer object can still suffer from local modifications - that occur at the Platform View. At this layer, an attacker can modify a process execution flow to extract information from the process engine, by adding one step that leaks Customer information. The Platform view is then impacted to ensure a valid execution flow that cannot be modified by external entities. The bottom layer, or the Infrastructure view is the place where we can certify the validity of the upper components. In our example, this bottom layer is used to ensure integrity of the process engine with a mechanism similar to a Trusted Platform Module.

Achieving these modifications from the bottom layer ensures that all components involved in integrity protection at higher layers cannot be further replaced and modified by unauthorized parties. We start from the infrastructure view to build trust on all layers. But the application designer who wants to implement its security property faces the complexity of modifying and configuring a huge amount of components and source code which he did not even author. Also, each layer has security concerns that differ from the other layers.

4.2 A solution to achieve Integrity with Aspects

A traditional approach to achieving integrity for such Customer objects requires the application designer to gather its developer team, security architects, business owners, and whoever is involved in application development, to configure and extend applications, modify programs, *etc.*. In this section, we provide some snippets of code that can, under certain conditions, make it possible to change the behavior of existing security services and components as well as business services, and also specialize services written in advance without the knowledge of a specific application through a separation of concerns between security and business requirements. This approach, which is more broadly discussed in the

next section, allows a fine-grained control of the implementation of requirements of an application that relies on a multi-layer SOA.

In the following, we illustrate with different code snippets how to address integrity in the impacted views. The snippets are independent from one another, that is they represent enforcement code to achieve a higher-level purpose. This purpose is driven by the architects and security experts that specify means guaranteeing properties at different levels. This entails the instantiation of invasive modifications as discussed in the previous section. The software layer provides a signature when receiving a Customer object from a collaboration. The platform layer provides process engine integrity, such that it guarantees the execution flow of a business process and prevents anybody from injecting a new step in the process that may leak a Customer object. Finally, the infrastructure layer provides means to guarantee trusted components at the platform level, by verifying that running components are those intended.

The aspects have to obey a precise lifecycle from specification via implementation to execution. We now consider the different roles involved in the correct application of aspects at the different levels. First of all, security aspects are created to respect specifications, given by a policy or by various requirements. We expect business owners to come with requirements that are refined by security experts. The aspect development per-se is done once with a high control on the code produce, thus the correctness. The development has to be handled by trusted developers, understanding security implication of their code in base-application. The deployment can be done prior execution by owners of the application, and execution is launched by platform administrator. The platform shall prevent consumers from modifying binaries and processes.

The first snippet in Listing 1.1 represents an aspect module that can be used to track message signature upon reception. It relates to standards such as WS-Security but with adaptation: our aspect code does not look for the systematic presence of signature tokens, but rather adds this specific mechanism when a Customer object is involved. It means that the client, prior to sending, adds a signature token when involving such an object. This behavior can be represented as a policy that clients understand and apply. The code shown is executed on the server side. Line 4 shows an annotation indicating the usage of this advice code for software view integrity. The pointcut at Line 5 uses a custom token *from-message*, which binds a message received by the application to the advice code to further investigate it. Lines 6 to 8 express the advice signature, which gets a Message and Customer object. Lines 10 to 22 contain the identification extraction from the message to verify the message signature. A valid signature indicates that the message has not been tampered with and can then be processed normally by the application.

```
1 @Aspect ("perProcess", SecurityProperty.INTEGRITY)
2 class CustomerIntegrity {
3   /* (i) Retrieve and validate sign info */
4   @Coverage(Level.SOFTWARE)
```

```
5   @Around("execution (* *(Customer)) && args(c) && from-
        message(m)")
6   public Object validateCustomerSignature (JointPoint jp
7       , Customer c
8       , Message m) throws CustomerIntegrityException{
9     //get public key of issuer
10    Identity issuer = m.getIssuer(); //retrieve identity
          from msg if present
11    X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec
          (Security.getPubKey(issuer));
12    KeyFactory keyFactory = KeyFactory.getInstance("DSA",
          "SUN");
13    PublicKey pubKey = keyFactory.generatePublic(
          pubKeySpec);
14    //get message signature
15    Signature sig = Signature.getInstance("SHA1withDSA", "
          SUN");
16    sig.initVerify(pubKey);
17    sig.update(m.getData());
18    //verify
19    boolean verifies = sig.verify(m.getInlineSignature());
20    if (!verifies)
21      throw new CustomerIntegrityException (c);
22    return jp.proceed();
23  }
24 }
```

Listing 1.1. Introducing Integrity at the Software Level

The second snippet in Listing 1.2 provides process engine integrity for business processes involving Customer manipulation. The underlying engine is an orchestration engine (BPMN 2.0 or BPEL engine) that should not be modified by unauthorized parties. More precisely, we allow activity and task execution only if the issuer is authorized to do so at a given stage in a business process. Lines 4 to 8 are a specific pointcut that respects AspectJ syntax - whenever an Activity execution is detected, a check is made to know if the process definition use a Customer object, thanks to a helper class. Then, at joinpoints, the advice described in Lines 10 to 20 is triggered. The behavior introduced is to check the prior execution of the activity and whether the action is authorized or not. The overall aspect allows a cross-cutting verification that business processes manipulating Customer objects are safe with regards to the planned behavior. It does not replace the engine enforcement but add an additional layer of confidence.

```
1 @Aspect ("perProcess", SecurityProperty.INTEGRITY)
2 class ProcessEngineFlowIntegrity {
3
4   @Pointcut("call(* *.*(Activity)) && args(a) && if()")
5   public static boolean processWithCustomer(Activity a) {
```

```
6     ProcessDefinition pd = a.getProcessDefinition();
7     return Helper.processUses(pd, Customer.class);
8   }
9
10  @Coverage(Level.PLATFORM)
11  @Around("processWithCustomer(Activity)")
12  public Object validateCustomerSignature (JointPoint jp
13      , Activity a) throws ProcessIntegrityException{
14    ProcessDefinition pd = a.getProcessDefinition();
15    for (User authorizedid : pd.getAuthorizedId(a){
16      if (a.getIssuer().equals(authorizedid))
17        return jp.proceed();
18    }
19    throws new ProcessIntegrityException(a);
20  }
21 }
```

Listing 1.2. Platform Level Integrity to control business process execution

The Listing 1.3 represents the behavior to be launched once per process execution. Prior to its execution, a process performs a check to verify the integrity of all involved components running in the infrastructure. Line 4 depicts an annotation that expresses the level related to views expressed in Figure 1. It means that the given code to be executed allows integrity coverage for the infrastructure level. Line 5 contains a custom pointcut element. The *start* expresses that whenever a Platform is used, the given advice has to be executed. Lines 6 to 8 represent the advice signature. We assume that we get an object called *SecurityPlatform* representing a security instance for the running process. Then, this object is used in Lines 9 and following to verify the integrity of the given platform being started. Deploying such code assumes that a valid Trusted Platform Module is present in the infrastructure view of our application, and that all components are correctly registered. The advice code ensures then that the process engine, among other platform components, has not been tampered with.

```
1 @Aspect ("perProcess", SecurityProperty.INTEGRITY)
2 class PlatformComponentsIntegrity {
3   @Coverage(Level.INFRASTRUCTURE)
4   @Before("start(Platform) && args(p)")
5   public void testPlatformIntegrity (JointPoint jp
6       , SecurityPlatform sp
7       , Platform p) throws PlatformIntegrityException{
8     sp.getTPM().verify(p.getTPMKeys());
9
10    if (sp & SecurityPlatform.CORRUPTED)
11      throw new PlatformIntegrityException(p);
12    if (sp & (SecurityPlatform.LOADING
13          | SecurityPlatform.CHECKING))
14      p.log ("Still waiting validation");
```

```
15      else
16        sp.validate(p);
17    }
18  }
```

Listing 1.3. Introducing Integrity at the Infrastructure Level

We have highlighted how the evolution of a security requirement impacts overall the different views involved in SOA execution. The snippets presented use AspectJ-like syntax enhanced with custom elements indicate how they are to be implemented in our framework. The pointcut language enrichment is one key to provide an easier mapping between the application execution and the overall security wanted, this is also under study. With the case of integrity, where one can request to protect integrity of specific data, we have shown it imply invasive modifications that pervade applications and components. Furthermore, components between them have no specific relation, due to the representation at the programming level. Nevertheless, aspect definition and application respect a global vision controlled by architects and business owners. We propose to develop a solution based on aspects that target not only software view of an application, but also platform and infrastructure view to achieve an overall application of integrity. Therefore a major research question relies on how the service model and the aspect model need to be designed to facilitate the relationship of non-functional concerns to the architectural components at each layer. Another constraint is the impact on the performance of the system. One research direction is to evaluate how load-time weaving or static weaving can help to reduce the overhead.

4.3 Aspect Model Design Criteria

In order to apply AOP to the evolution of security requirements defined according to the different views we have, aspects have to meet basic properties. Generally, aspect models come in very different forms, concerning their basic concepts but also implementation strategies, suitability for the application of formal methods, etc. The proposed aspect model that is used in the previous section has different characteristics. We presented some in snippets code (Listings 1.1,1.2,1.3) but try to formalize our contribution in this section - a work which is currently in progress.

The aspect model we envision is based on the pointcut-advice model for aspects, with some important extensions to be applied. The pointcut-advice model is characterized by three main abstractions: aspects, pointcuts and advice that together provide means for the concise definition and efficient implementation of so-called crosscutting functionalities of a base application, such as security, that cannot typically be modularized with existing structuring and encapsulation mechanisms, such as services or components. We address these requirements by the following set of major characteristics that the aspect model has to fulfill. These characteristics are for most of them general in the sense that they apply

to all three basic aspect abstractions (aspects, pointcuts and advice) - except if stated otherwise in the following:

- ***Basic abstractions and relations***: The pointcut language should enable referencing all relevant abstractions of the service model and the concrete infrastructures; the advice language allows to manipulate these entities. Concrete examples for such abstractions include collaborations, processes, services and resources. Relevant relationships between them include relations between adjacent abstraction levels or the ability to protect some of them using certain security mechanisms, such as access control, while others may not be modified by that security mechanism.
- ***Composition model***: The aspect model should provide a gray-box composition model, i.e., aspects may access parts of service implementations. However, such access can be restricted by explicit fine-grained conditions on the structure and behavior of the underlying base system. The aspect model will therefore provide strong control over invasive composition. Corresponding conditions will be defined as part of evolution tasks through the aspects that realize them. The conditions may then be integrated before execution in the runtime representations of aspects or the underlying infrastructure, or enforced, possibly at execution time, on service implementations.
- ***Dynamic application***: Aspects should be applicable dynamically even though static application strategies may also be used, especially for the introduction of security mechanisms that would suffer from an excessive overhead. Many current aspect models only support a static or load-time application of aspects, which severely limits their applicability for many composition tasks. Our model therefore significantly broadens the use of aspects to many real-world scenarios that involve highly dynamic service applications. Another general characteristic of our model is that the model enables the aspect-based definition of service evolutions whose (security) properties can be formally analyzed.
- ***Formal properties***: The aspect model should include explicit means to restrict aspects, pointcuts and advice, such that relevant formal properties of service evolutions defined using aspects can be specified precisely, formally analyzed and enforced on corresponding implementations.
- ***Protocol support***: The pointcut language should include direct support for matching (parts of) protocols that govern the collaboration (choreography etc.) between entities of the service model. The advice language permits the manipulation of protocols.
- ***Local state***: Aspects may contain local state that can be used to modify state of the base application. Aspect definitions may, however, restrict the kind of state that can be defined and used.

Even though our solution is not fully developed yet, we highlighted the need for a new approach to secure not only classical layers (software view) but the entire chain of components supporting service processing and coordination. Aspect-based techniques meet our requirements to express and apply concerns that are

normally difficult to address as their impact is scattered in a multi-layer SOA stack.

5 Related Work

Service-oriented systems are characterized by complex interactions among functional, management and infrastructure interfaces. Aspect-oriented approaches have been proposed to address such issues, although only in the context of existing orchestration services for SOAs.

Aspect-Oriented Software Development (AOSD) [8,1] has emerged over the previous decade as the domain of systematic exploration of crosscutting concerns and corresponding support throughout the software development process. AOSD has been applied to service-oriented systems (mainly based on web services) and also to the modularization of crosscutting security policies for sequential and distributed systems (including few work on services). However, current AOSD approaches typically target a single virtual machine or platform and the process of coordinating aspects through points of execution is centralized. Some recent approaches, such as AWED [3] and DyMac [9] support the modularization of the crosscutting functionalities of distributed applications. These systems extend basic concepts of AOSD such as pointcuts and advices to remote pointcuts and remote advices that allow concerns to be composed across different locations.

The specific case of security in service-oriented architectures has also been addressed through aspect-oriented approaches. For instance, Courbis and Finkelstein [6] proposed to weave aspects into web service orchestrations. Service orchestrations, in particular using the language BPEL, have also been extended with aspect support by Charfi and Mezini [4]. Still, not much has been done to date to provide support for multilevel horizontal and vertical service compositions or for enforcing security mechanisms at multilevel service composition in the presence of aspects. Furthermore, in a distributed context, and specifically in service-oriented computing, there are currently still very few results on the enforcement and preservation of security properties in the presence of aspects. Svirskas et al. [16] have proposed a mechanism of structured compliance proof that guarantees that these protocols are enacted in compliance with the effective policies and regulations. However, these approaches are only considering service compositions at one level.

As to the evolution of SOAs, Chen et al. [5] have proposed an extensible SOA-based platform and provided a roadmap for SOA evolution. Mingyan and Yanzhang [13] presented a service-oriented dynamic evolution model named SOEM model and gave a formal description of a series of concept in service-oriented software evolution process. Several research challenges in maintenance and evolution of SOA are also discussed in [10], like for instance multilanguage system analysis and maintenance, the reengineering of processes for migration to SOA environments and evolution patterns of Service-Oriented Systems. However, these development methods and techniques for SOA lack means to analyze

and preserve properties in the presence of evolution, especially regarding security properties.

6 Conclusion and Future Work

Applications in service-oriented computing have traditionally been developed using composition in homogeneous and simple frameworks. However, the trend towards increasingly complex infrastructures indicates a need for a general service model that would make it possible to achieve a uniform approach yet to separate business process concerns from their diverse security requirements (which are even more subject to evolution over time). SOA cannot rely on the only composition of services and disregard the security of other software layers on top of which services are implemented.

We have exposed in this paper in what respect the complexity of service-oriented architectures and their evolution involve challenging research problems, in particular with respect to the understanding of security threats and even more so to the specification of security properties and mechanisms.

Research Challenges: We can summarize the major challenges we will address in the near future in the following questions:

- How to create appropriate abstractions for the different architectural layers in SOA's as to understand the impact of evolutions to the overall system security?
- How to provide support for threat analysis in cross-domain multi-layered SOAs, when moving towards to deployment models based on clouds and integration of mobile devices?
- How security concerns can be modularized and uniformly deployed in cross-domain service orchestrations using vertical and horizontal aspect orientation capabilities? We also discussed in this paper a new approach to the description of the service model relying on invasive aspects introduced at multiple levels of abstraction. This model will make it possible to develop and evolve service-oriented applications in dynamic business contexts, while keeping the complexity of managing their security manageable.
- How to manage possibly conflicting security requirements, specifically when composing distinct security aspects at a given level? We have started defining a formal model for secure services in order to deal with possibly conflicting security requirements. This specification will play a central role in the design of a flexible aspect model appropriate for dealing with multi-layer security concerns. We shall investigate how to provide security requirements specifications such that these conflicts can be detected and solved.

Acknowledgment

This work was supported by the ANR, the French National Research Organization through the project CESSA (Compositional Evolution of Secure Services with Aspects, ID.: 09-SEGI-002-01).

References

1. M. Akşit, S. Clarke, T. Elrad, and R. E. Filman, editors. *Aspect-Oriented Software Development*. Addison-Wesley Professional, Sept. 2004.
2. E. Bagheri and A. Ghorbani. A service oriented approach to critical infrastructure modeling. *In Workshop on Service Oriented Techniques. National Research Council, Canada*, 2006.
3. L. D. Benavides Navarro, M. Südholt, W. Vanderperren, and B. Verheecke. Modularization of distributed web services using awed. In *Proc. of the th Int. Conf. on Distributed Objects and Applications (DOA'06*, volume 4276 of *LNCS*, pages 1449–1466. Springer Verlag, Oct. 2006.
4. A. Charfi and M. Mezini. Ao4bpel: An aspect-oriented extension to bpel. *World Wide Web*, 10(3):309–344, 2007.
5. Q. Chen, J. Shen, Y. Dong, J. Dai, and W. Xu. Building a collaborative manufacturing system on an extensible soa-based platform. In *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, pages 1 –6, may. 2006.
6. C. Courbis and A. Finkelstein. Weaving aspects into web service orchestrations. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 219–226, Washington, DC, USA, 2005. IEEE Computer Society.
7. M. S. Idrees, G. Serme, Y. Roudier, et al. State of the art and requirement analysis of security functionalities for soas. Deliverable D2.1, The CESSA project, July 2010. `http://cessa.gforge.inria.fr/lib/exe/fetch.php?media=publications:d2-1.pdf`.
8. G. Kiczales. Aspect-oriented programming. *ACM Comput. Surv.*, 28(4es):154, 1996.
9. B. Lagaisse and W. Joosen. True and transparent distributed composition of aspect-components. In *In Proc. Middleware'06*, pages 42–61. Springer, 2006.
10. G. Lewis and D. Smith. Service-oriented architecture and its implications for software maintenance and evolution. In *Frontiers of Software Maintenance, 2008. FoSM 2008.*, pages 1 –10, sep. 2008.
11. L. Lowis and R. Accorsi. On a classification approach for soa vulnerabilities. In *International Computer Software and Applications Conference*, pages 439–444, 2009.
12. L. Lowis and R. Accorsi. Vulnerability analysis in soa-based business processes. *IEEE Transactions on Services Computing*, 99(PrePrints), 2010.
13. Z. Mingyan, W. Yanzhang, C. Xiaodong, and X. Kai. Service-oriented dynamic evolution model. In *Computational Intelligence and Design, 2008. ISCID '08. International Symposium on*, volume 1, pages 322 –326, oct. 2008.
14. OWASP. Open web application security project https://www.owasp.org/index.php/category:attack.
15. G. Serme, M. S. Idrees, Y. Roudier, et al. Compositional evolution of secure services using aspects. Deliverable D3.1, The CESSA project, July 2011. `http://cessa.gforge.inria.fr/lib/exe/fetch.php?media=publications:d3-1.pdf`.
16. A. Svirskas, J. Isacenkova, and R. Molva. Towards secure and trusted collaboration environment for European public sector. In *TrustCol 2007, 2nd International Workshop on Trusted Collaboration, November 12th-15th, 2007, New York, USA*, 11 2007.