# The Service Security Lab: A Model-Driven Platform to Compose and Explore Service Security in the Cloud

Michael Menzel, Robert Warschofsky, Ivonne Thomas, Christian Willems, Christoph Meinel
*Hasso-Plattner-Institute*
*Prof.-Dr.-Helmert Str. 2-3*
*14482 Potsdam, Germany*
{michael.menzel, robert.warschofsky, ivonne.thomas, christian.willems, meinel}
@hpi.uni-potsdam.de

## Abstract

*Cloud computing enables the provisioning of dynamically scalable resources as a service. Next to cloud computing, the paradigm of Service-oriented Architectures emerged to facilitate the provisioning of functionality as services. While both concepts are complementary, their combination enables the flexible provisioning and consumption of independently scalable services. These approaches come along with new security risks that require the usage of identity and access management solutions and information protection. The requirements concerning security mechanisms, protocols and options are stated in security policies that configure the interaction between services and clients in a system.*

*In this paper, we present our cloud-based Service Security Lab that supports the on-demand creation and orchestration of composed applications and services. Our cloud platform enables the testing, monitoring and analysis of Web Services regarding different security configurations, concepts and infrastructure components. Since security policies are hard to understand and even harder to codify, we foster a model-driven approach to simplify the creation of security configurations. Our model-driven approach enables the definition of security requirements at the modelling layer and facilitates a transformation based on security configuration patterns.*

## 1. Introduction

The concept of cloud computing facilitates the idea of providing dynamically scalable resources as a service on the Internet. As a way to structure the resources provided by cloud computing, a distinction in several layers can be used [1, 2]. Infrastructure as a Service (IaaS) is considered as the lowest layer and provides virtualised computer infrastructure components as a service. Above this, Platform as a Service (PaaS) delivers computing environments tailored to specific needs as a service. Service access or integration is enabled by the Components as a Service (CaaS). The upper-most layer, Software as a Service (SaaS), provides complex software functionality.

Next to cloud computing, Service-oriented architectures are based on the idea of exposing software functionality as services to be used by independent parties. Their inherent independence of a specific platform and operating system make them perfectly suitable to connect service consumers and service providers over the Internet and provide a technical foundation for cloud computing as outlined in [2]. Dawoud et al. [1] described SOA and cloud computing as complementary concepts that intersect in the CaaS layer providing e.g. Web Services. In addition, a platform can be offered at the PaaS layer to provide custom Web Services. These services can be used by composed applications (offered as SaaS) that orchestrate and consume services and data from different sources. The flexibility concerning the integration of resources and the independent scalability of each service are the major benefits of the combination of both concepts.

However, the combination of SOA and cloud computing facilitating the provision of composed application and services that integrate and orchestrate services from different sources pose new challenges to security. Since services and applications are exposed to the Internet and are used in a global context, the management of user identities across organisational borders is a key element to perform access control and to prevent unauthorised access in a decentralised environment. Open Identity Management Models support the sharing of identity information across several trust domains in a controlled

manner. Clients can request identity information from the identity management systems and convey this information in an interoperable format to a requesting party.

Besides identity provisioning, confidentiality and integrity of exchanged, stored, and processed information must be ensured. Several specifications emerged to protect information at different layers. For instance, a secure channel can be used to protect exchanged information, while signature and encryption mechanisms applied to a message can also protect stored and processed information.

In general, these security requirements are stated in security policies that configure the secure interaction of participants in a service-based system. Policies facilitate the negotiation of security requirements between services and service clients to enable interoperability at runtime. This enables a seamless usage of services in the cloud to build composed applications.

However, due to the complexity of the involved specifications, the variety of security mechanisms and the flexibility of service-based systems, such policies are hard to understand and even harder to codify. To overcome these limitations, we foster a model-driven approach that generates security configurations based on system design models annotated with security requirements. Our model-driven approach integrates security intentions in system design models using the integration schema introduced by SecureUML in [3] and enables a modeller to state basic requirements on an easily comprehendible level. However, the transformation from simple security intentions to complex security configurations is a challenging task, since different strategies might exist to enforce a security intention. Since security expert knowledge is required to perform this transformation, we have specified and formalised a security pattern system for service-based systems that provides this knowledge. This pattern system is used to guide and automate the transformation process from modelled security intentions to security configurations.

Our model-driven approach constitutes the foundation for our Service Security Lab that facilitates the provision, consumption, and orchestration of cloud-based Web Services. These services are integrated in composed applications that can be created by the user and are instantiated in a virtualised environment. Our cloud platform facilitates developers and security engineers to test Web Services with different security configurations and in different security environments. The functioning of security modules can be monitored and analysed as well as the enforcement and application of security mechanisms. Our model-driven approach enables a simple and comprehensible generation of these security configurations.

Altogether, we present our Service Security Lab in this paper that

- enables a user to create Web Service-based composed web applications using a visual modeller.
- enables a user to upload, orchestrate, execute, and test own services.
- secures a composed application using a pattern-driven generation of security configurations.
- enables to monitor communication and analyse the security modules used to enforce security.

This paper is structured as follows. Section 2 introduces basic security concepts to secure services in the cloud. Our model-driven approach to setup and configure secure systems is explained in the next Section. Section 4 provides an overview about the architecture of our Service Security Lab. Building on that, Section 5 outlines the usage steps of our platform to instantiate and execute a secured system. Related work is discussed in Section 6, while Section 7 concludes this paper and presents future work.

## 2. Securing Services in the Cloud

This section gives a brief introduction into security concepts required to implement security in decentralised systems as SOA and the cloud.

### 2.1. Identity and Trust Management

Traditionally used identity management solutions were designed for closed domains. User attributes were stored in a proprietary data model and exchanged in a fixed format between the identity management and applications. While participants within the domain understand this format, this would not apply for those outside the domain. Open identity management models rely on interoperable systems to easily express, intuitively select and securely transport identity information. As no single identity system is suitable for every identity scenario, an identity metasystem has been specified e.g. with the OASIS Identity Metasystem Interoperability specification 1.0 [4]. It abstracts from concrete identity management technologies and provides the necessary mechanisms to describe, exchange and distribute identity information across identity management solutions. Usually, three different roles are considered in the Identity Metasystem as depicted in Fig. 1.

- The Subject represents the user to which the digital identity is associated.

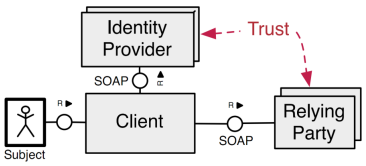


**Figure 1. Identity Metasystem Participants**

- An Identity Provider supplies the digital identity for the Subject. In order to request identity information, an Identity Provider offers a Security Token Service (STS).
- A typical Relying Party is an application which relies on the issued security tokens.

The architecture of the identity metasystem is based on open standards such as WS-Trust. One of the main problems inherent to open identity management models is the establishment of trust relationships between independent parties. In order to trust on external security tokens, an established trust relationship is a necessary prerequisite.

### 2.2. Policy Management

Security requirements expressed in policies comply to security goals such as confidentiality, integrity, authentication and authorisation. To fulfill these requirements, the service consumer may use for instance WS-Security to protect exchanged information and to convey security tokens that provide identity information.

In general, the standards WS-Policy and WS-SecurityPolicy are used to express security requirements of services in a machine-readable way. WS-Policy provides a general framework for web service policies, while WS-SecurityPolicy is an extension with elements to express security related requirements.

To invoke a service, a service consumer can request such a policy from the service. The consumer intersects this policy with its own policy to find a policy alternative that fits both – service and consumer. Using the selected policy alternative, the consumer can request the service. The service must reject incoming messages that do not meet the requirements stated in its policy.

## 3. Model-driven Creation of Secure Composed Applications

Our cloud-based Service Security Lab facilitates users to orchestrate, execute and test composed applications using a visual modeller. To implement the functional and security requirements specified at the modelling layer, our cloud platform has to ensure two aspects: The system with all involved services and web application components must be instantiated in a virtual machine according to the functional requirements and the services must be configured in compliance with the modelled security requirements. As illustrated in Fig. 2, our approach consists of three layers. Functional and security requirements, expressed at the modelling layer, are translated to a platform independent model. This model constitutes the foundation to setup the virtual machine, application server, services and composed applications that are provided to the user.

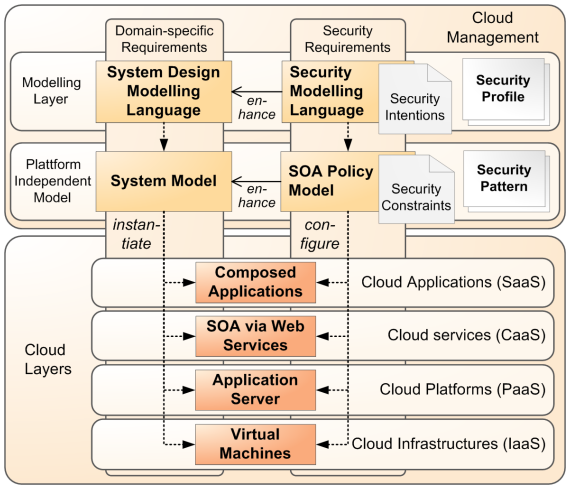


**Figure 2. Model-driven Security in SOA**

In particular, our model-driven approach requires an automated generation of enforceable security configurations based on the modelled security requirements. The modelling of these requirements, the structure of the platform-independent model and the transformation process across these layers are explained in this Section. Additional information concerning this model and our pattern-driven transformation process are provided in [5] and [6].

### 3.1. Modeling Secured Systems

System design models such as FMC block diagrams or UML models are the foundation to enable users to model the structure of a system and to state security requirements in an easy accessible way. The elements in these modelling languages are annotated with security intentions that are defined by our security modelling language SecureSOA [6]. SecureSOA uses the integration schema defined by SecureUML [3] to enable the enhancement of system design models with security-related modelling elements.

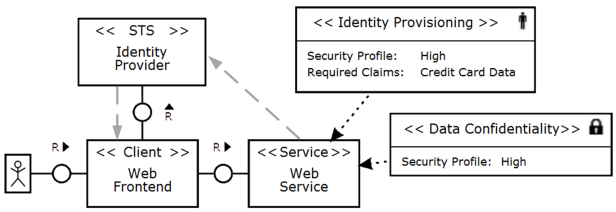


**Figure 3. Modelling Security Intentions**

A simple example diagram is shown in Fig. 3. A user leverages a web frontend to access a service. Moreover, a Security Token Service (STS) is deployed that is trusted by the service and the user. The STS can authenticate the user and issue a security token. These tokens can be sent along with the request message to access the service.

In addition to the system structure, Fig. 3 depicts two security intentions representing security goals that must be enforced by the security infrastructure. Each security intention refers to a security profile that lists security mechanisms that can be used by our platform to implement security in an instantiated use case. Profiles are used to abstract from technical details at the modelling layer. They are predefined by our platform but can be adjusted while configuring a use case to meet the user needs. The profile *High* for Identity Provisioning in Fig. 3 requires the usage of SAML tokens, while the *Low* profile would allow Username and Password.

## 3.2. A Platform-independent Model for Secure Service-based Systems

As aforementioned, we use a domain-independent model as an abstraction layer to concrete security configurations. Our model describes basic entities in a service-based system (e.g. clients and services), the relationships and interactions between these entities, and the information that is exchanged in the scope of these interactions.

In addition, a policy meta-model is part of our domain-independent model and supports the expression of security requirements concerning communication related security goals. Our model serves as an abstraction layer for security policy languages and simplifies the handling and statement of security requirements.

A policy in our platform-independent policy meta-model consists of several *Policy Alternatives* that contains a list of *Security Constraints*. In general, a *Security Constraint* describes a requirement to fulfill a *Security Goal* and contains information describing what has to be secured and which security mechanisms must be used. An example is shown in Fig. 4. The *User Authentication Constraint* requires that the sender of a message adds information about his identity to the message. Therefore, this constraint references a list of required claims and an issuer of the identity information.

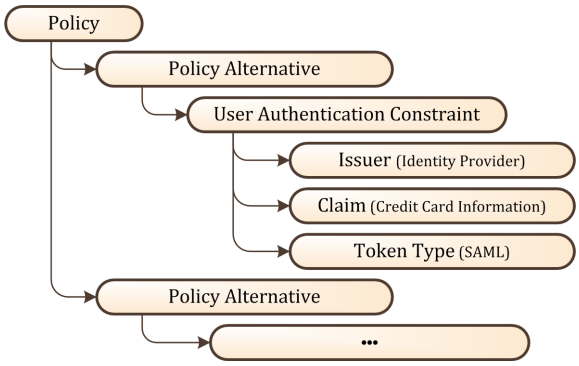


**Figure 4. Policy Model Exmaple**

## 3.3. A Model-driven Transformation

The transformation of a system design model to a secured running system is performed as follows: information specified at the modelling layer is gathered in our domain-independent model. Finally, the information in the model is transformed to configuration files that are used to instantiate the system.

**3.3.1. Pattern-based Transformation of System Design Diagrams.** Two types of information have to be extracted from the modelled system diagrams and must be transformed to our domain-independent model: Information about the structure of a system (e.g. involved entities and their relations) and security related aspects. The transformation of the system structure visualised in a system diagram to our domain-independent model is straightforward, since these modelling elements (e.g. a service modelled in FMC) can be mapped directly.

However, the transformation from abstract security intentions to security constraints for our policy model is quite challenging, since a simple mapping is not sufficient. Expert knowledge might be required to determine an appropriate strategy to secure services and resource, since multiple solutions might exists to satisfy a security goal. For example, confidentiality can be implemented by securing a channel using SSL or by securing parts of transferred messages using WS-Security. To describe these strategies and their preconditions in a standardised way, we foster the usage of security configuration patterns that are based on the idea of design patterns [7]. We have defined a formalised system of security configuration patterns to enable an automated application of security patterns in the transformation process. Each

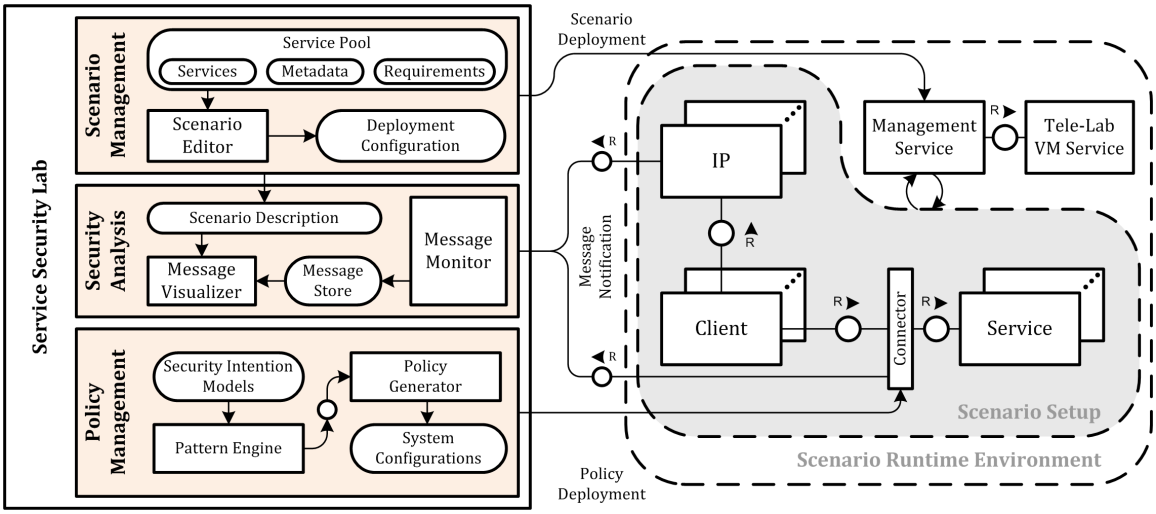


**Figure 5. Service Security Lab Architecture**

security configuration pattern provides a set of policy constraints (*solution*) for a security intention (*problem*). In addition, conditions are defined for each pattern that determine its applicability.

Overall, our security configuration patterns enable an automated creation of security policy constraints based on simple security intentions [5].

**3.3.2. Generating System Configurations.** The final step in our model-driven approach is the transformation of security configurations into enforceable WS-SecurityPolicies. Thereby, the *Policy Alternatives* are transformed separately into alternatives of WS-Policy containing required WS-SecurityPolicy assertions. The transformation of an alternative is divided into three phases to generate the binding assertion, the protection assertions, and the supporting token assertions of the alternative. These assertions describe requirements regarding the usage of WS-Security and WS-Trust in terms of the provision of security tokens and the use of encryption and signature algorithms and options [8].

## 4. The Service Security Lab

Our Service Security Lab is a cloud platform that enables users to assemble and test secure service-based applications. These composed applications are created using our model-driven approach described in the previous section. Fig. 5 gives an overview about the components in our architecture. Our platform consists of three main components that realise the dynamic instantiation of composed applications, creation and distribution of security configurations, as well as the monitoring and analysis of applied security mechanisms.

1. **Scenario Management** – The scenario management component enables users to model the system structure in a visual designer. Services provided by a repository (service pool) can be mapped to the services modelled in the system diagram. In addition, this component is responsible to setup and configure a composed application in a virtual machine in accordance to a model. The management service creates a copy of a virtual machine image that provides a basic system with an application server and a management web service. The virtual machine's management web service is used to deploy services and configuration files within the virtual machine as shown in Fig. 5. The Service Security Lab provides a fresh and exclusive virtual machine for each user session. Automatic VM provisioning and rollback after usage is realised using the Tele-Lab virtual machine management services. Tele-Lab [9] is a virtual lab intentionally developed for providing a hands-on training environment for IT security lections, but can dynamically provide virtual machines from preconfigured images for any purpose.

2. **Policy Management** – In addition to the system structure, the visual designer can be used to state security intentions to specify security requirements. The security management component generates system configurations based on the modelled security requirements using the transformation process described in Section 3.3. The pattern engine shown in Fig. 5 performs the first transformation step using our security configuration pattern system and resolves a platform-independent

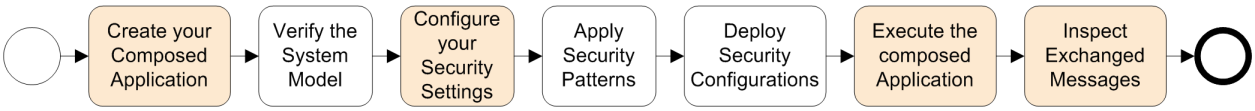


**Figure 6. Process to Configure and Execute a Composed Application**

system model and security constraints. In the second transformation step, concrete system configurations (e.g. WS-SecurityPolicy) are created by the policy generator component. The created policies and configuration files are used by the scenario management component to configure the composed application.

3. **Security Analysis** – After executing a use case, a user can analyse the exchanged messages and inspect security modules. This is done by the security analysis component and is based on monitoring agents deployed in the virtual machine. These agents intercept messages and forward these messages to the message monitor of the security analysis component. The message visualiser aligns monitored messages with the modelled entities and enables users to track the communication and the behaviour of security modules easily.

## 5. Demonstration

This section illustrates the process that is performed to setup, instantiate and run a composite application using our cloud platform. Moreover, the modelling of scenarios is described in detail, since this is the foundation to create secure composite applications.

### 5.1. Usage Steps

Fig. 6 illustrates the overall process and highlights activities that are performed by the user:

**Model your Composed Application** – In the first step, the user can create a composed application by modelling the structure of the desired system. Moreover, security related aspects such as security intentions and trust relationships can be modelled as well. For this purpose, we integrated the web-based modelling tool Oryx [10] in our platform and added our security design language SecureSOA (see Section 3.1) as a stencil set for Oryx. Scenario modelling is illustrated in Section 5.2 using the example of a web shop scenario.

**System Model Verification** – The cloud platform verifies the modelled system to ensure that an executable system can be created. Moreover, the modelled security aspects are verified to ensure that security patterns can be applied to secure the system. For instance, the security intention *user authentication* requires a trust relationship between the service and its clients, since these users must be registered at the service or at an identity provider to enable the authentication.

**Configure your Security Settings** – In addition to the system model, the user can specify security profiles that are referenced by the security intentions. As introduced in Section 3.1, each profile describes security mechanisms that can be used to secure the composite application.

**Apply Security Patterns** – Security patterns describe strategies to secure a service (Section 3.3). Our security configuration patterns resolve a set of security mechanisms and protocols for each component in the system. These requirements are stored in our platform independent policy model as security constraints.

**Create Security Configurations** – Our platform independent model is transformed to concrete security policy languages (e.g. WS-SecurityPolicy in the scope of Web Services) that can be deployed and enforced at the services and frontends used in the composed application.

**Execute the Composed Application** – The Tele-Lab VM management (Fig. 5) provides a virtual machine for the user. The applications and security policies created in the previous step are deployed to the VM by the scenario management component. The user can access the scenario using a web frontend.

**Inspect Security Modules and Exchanged Messages** Our platform enables users to gain insight into services and the security modules used to enforce security policies. For each service, the security modules can be visualised that process and secure requests and responses. The messages that passed these modules can be inspected as well. Fig. 7 shows the visualisation in our platform that depicts a chain of security modules and a service request

that passed this chain. The message security protocols and mechanisms used to secure this message are analysed and highlighted.

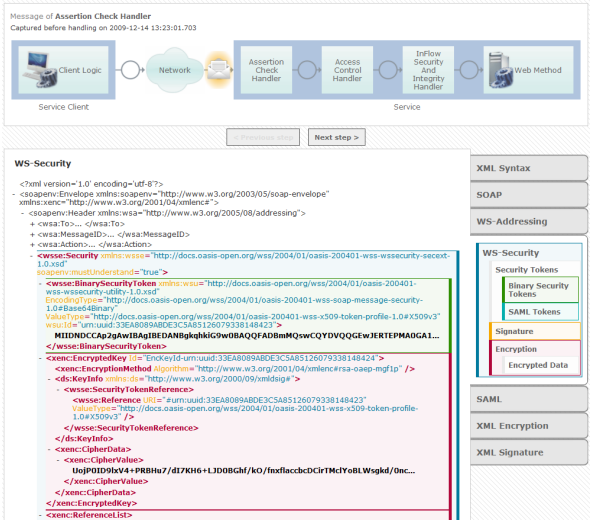


**Figure 7. Security Module and Message Visualisation**

## 5.2. Modelling a Scenario

Fig. 8 illustrates the structure of a composite example application and its notation using the visual modeller that is integrated into our Service Security Lab. This scenario defines an online store web application that uses three external services; a catalogue, a payment and a shipping service. Note that each modelled entity indicates its role (Client, Service or STS) using UML stereotypes. The catalogue service implements an interface to list and query products that are offered by the online store. This modelled entity is mapped to a web service listed in the service repository that integrates Amazon web services. The payment service shown in Fig. 8 represents an external service which handles the payment for the store. In order to do so, the service needs payment information including a payment amount and credit card data. Moreover, a shipping service is required by the front end that initiates the shipping of the goods using the recipients address.

Users in this scenario have an account at their trusted bank and at the registration office, who act as identity providers managing the user's digital identity as introduced in Section 2.1. The payment and the shipping service have established a trust relationship with these identity providers.

In addition, various entities in this use case are annotated with security intentions. The intentions data

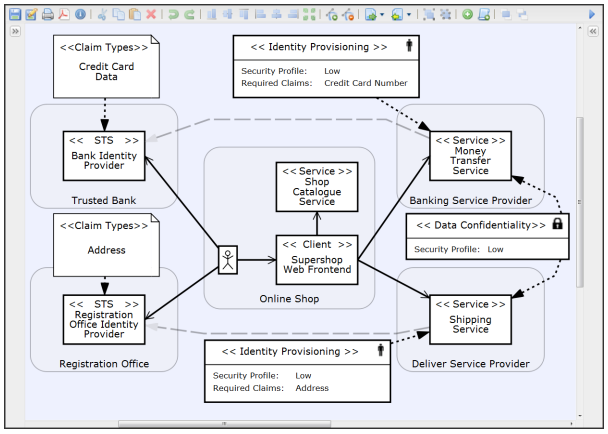


**Figure 8. SecureSOA Web Shop Example**

confidentiality and identity provisioning are used to ensure the protection of the exchanged messages and to require the provisioning of identity information.

## 6. Related Work

Modelling security requirements for secure systems is an emerging topic. Rodríguez *et al.* [11], Wolter [12], and Breu *et al.* [13] proposed enhancements for process models and UML diagrams to express security requirements. However, these approaches do not provide a transformation of communication-related security requirements to enforceable security policies such as WS-SecurityPolicy.

This has been addressed by Jensen and Feja who described a model-driven generation of Web Service security policies to enforce data protection [14].

SecureUML [3] introduced by Basin et al. is a security modelling language to describe role-based access control and authorisation constraints. To integrate this language in different types of system design languages, they proposed an integration schema that is the foundation of the modelling approach used in this paper.

Jürjens presented UMLSec [15] to express and verify security relevant information within UML-diagrams. Since all security aspects need to be described at the modelling layer, this approach does not provide a simple notion for security intentions.

Satoh and Yamaguchi introduce an intermediate model to transform a WS-SecurityPolicy into platform-specific configuration files [16]. This model represents the WS-Security message structure and the meanings of signatures and encryption specified in a WS-SecurityPolicy.

In the recent years, various security patterns have been defined. Using these security patterns, Delessy de-

scribed a pattern-driven process for secure SOAs [17]. An automated generation of security policies is not described.

## 7. Conclusion and Future Work

In this paper, we presented our cloud-based Service Security Lab that enables the creation and on-demand provisioning of composed applications and Web Services orchestrations. This platform is designed as a virtualised testing environment for service-related security concepts and security components. Users can create composed application using a visual modeller that specifies the structure of the applications in terms of consumed and orchestrated services (e.g. data providing services). The user can integrate external services (e.g. Amazon services), use the cloud platform to provide his own services (Platform as a Service) or compose predefined services provided by the cloud platform (Component as a Service). These composed applications and services are instantiated and provisioned on-demand in a virtualised environment for each user. Our platform enables users to run these applications and to monitor the security components in the system. A user can specify security requirements for each service and can analyse how security requirements are enforced.

However, the generation of security configurations in a distributed system is a complex task, since various security mechanisms and options can be chosen. To simplify the generation of security policies that can be applied to the orchestrated services, we introduced a model-driven approach that transforms security intentions to enforceable security policies. This transformation is based on a set of security configuration patterns that provide security expert knowledge to configure the system. Our platform allows users to specify these security intentions in system models to enable a simple and easily comprehensible specification of security requirements. While instantiating a composed application for a user, the services are configured according to the generated policies using our model-driven approach.

## References


[1] W. Dawoud, I. Takouna, and C. Meinel, "Infrastructure as a Service Security: Challenges and Solutions," *Proceedings of the 7th IEEE International Conference on Informatics and Systems*, p. 8, March 2010.

[2] M. Jensen, J. Schwenk, N. Gruschka, and L. L. Iacono, "On technical security issues in cloud computing," *Cloud Computing, IEEE International Conference on*, vol. 0, pp. 109–116, 2009.

[3] D. Basin, J. Doser, and T. Lodderstedt, "Model driven security: from uml models to access control infrastructures," *ACM Transactions on Software Engineering and Methodology*, vol. 15, no. 1, pp. 39–91, January 2006.

[4] OASIS, "Identity Metasystem Interoperability Version 1.0 ," *OASIS Standard*, July 2009. [Online]. Available: "http://docs.oasis-open.org/imi/identity/v1.0/identity.html"

[5] M. Menzel, R. Warschofsky, and C. Meinel, "A Pattern-driven Generation of Security Policies for Service-oriented Architectures," in *IEEE International Conference on Web Services (ICWS 2010)*, 2010.

[6] M. Menzel and C. Meinel, "SecureSOA - Modelling Security Requirements for Service-oriented Architectures," in *IEEE International Conference on Services Computing (SCC 2010)*, 2010.

[7] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobsen, I. Fiksdahl-King, and S. Angel, *A Pattern Lanuage: Towns - Buildings - Construction*. Oxford University Press, 1977.

[8] G. Della-Libera, M. Gudgin, and et all, "Web services security policy language (ws-securitypolicy)," Public Draft Specification, Juli 2005.

[9] C. Willems and C. Meinel, "Tele-lab it-security: an architecture for an online virtual it security lab," *International Journal of Online Engineering (iJOE)*, vol. 4, 2008.

[10] G. Decker, H. Overdick, and M. Weske, "Oryx - an open modeling platform for the bpm community," in *BPM*, 2008, pp. 382–385.

[11] A. Rodríguez, E. Fernández-Medina, and M. Piattini, "A bpmn extension for the modeling of security requirements in business processes," *IEICE Transactions*, vol. 90-D, no. 4, pp. 745–752, 2007.

[12] C. Wolter and A. Schaad, "Modeling of task-based authorization constraints in bpmn," in *BPM*, 2007, pp. 64–79.

[13] M. Hafner and R. Breu, *Security Engineering for Service-oriented Architectures*. Springer, October 2008.

[14] M. Jensen and S. Feja, "A security modeling approach for web-service-based business processes," *Engineering of Computer-Based Systems, IEEE International Conference on the*, vol. 0, pp. 340–347, 2009.

[15] J. Juerjens, "UMLsec: Extending UML for Secure Systems Development," in *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, 2002, pp. 412–425.

[16] F. Satoh and Y. Yamaguchi, "Generic security policy transformation framework for ws-security," in *IEEE International Conference on Web Services (ICWS 2007)*. Los Alamitos, CA, USA: IEEE Computer Society, 2007, pp. 513–520.

[17] N. A. Delessy, "A pattern-driven process for secure service-oriented applications," Ph.D. dissertation, Florida Atlantic University, Boca Raton, Florida, May 2008.