

# SecureSOA – Modelling Security Requirements for Service-oriented Architectures

Michael Menzel, Christoph Meinel

Hasso-Plattner-Institute,  
Prof.-Dr.-Helmert-Str. 2-3  
14482 Potsdam, Germany  
{michael.menzel, meinel}@hpi.uni-potsdam.de

## Abstract

*Service-oriented Architectures (SOA) facilitate the provision and orchestration of business services to enable a faster adoption to changing business demands. Web Services provide a technical foundation to realize this paradigm and support a variety of different security mechanisms and approaches. Security requirements are codified in Web Service policies that control the service's behavior in terms of secure interactions with other participants in an SOA. To facilitate and simplify the generation of enforceable security policies, we foster a model-driven approach based on the modelling of security requirements in system design models. This paper introduces our security design language SecureSOA that enables the definition of these security requirements. We present the abstract syntax and notion of SecureSOA and describe a schema to integrate SecureSOA in any system design language for service-based systems. Moreover, we will demonstrate the integration of SecureSOA in Fundamental Modelling Concept (FMC) Block Diagrams.*

## 1 Introduction

IT-infrastructure have evolved into distributed and loosely coupled service-based systems that are capable to expose company's assets and resources as business services. To implement this paradigm, the Web Service specification provide a technical foundation based on XML-messaging. In addition, Web Services facilitate the usage of different security patterns, mechanisms and algorithms to secure the interaction between the participants in an Service-oriented Architecture.

Security policies for SOA (e.g. WS-Policy and WS-

SecurityPolicy) are used to state these security requirements on a technical layer concerning the usage of specifications such as WS-Security. These policies enable services to communicate requirements to its service consumers, for example, to inform about required identity information, trusted parties or required mechanisms to secure exchanged information.

To enable a simplified generation of security policies, we foster a model-driven approach that integrates security intentions in SOA system models. SOA system models provide an abstract view on different aspects such as participants, information, and processes. The integration of security intentions enables a modeller to state basic requirements on a technically independent level. For instance, confidentiality requirements or required identity information can be annotated.

Modelling security has been a research topic in recent years. Jürjens introduced the UMLsec extension [1] to express and verify security aspects within UML-diagrams. However, to perform such a formal verification, all security-related aspects such as cryptographic data must be specified in the system model. This would be a suitable approach for security experts, but it tend to be difficult to understand without a strong security background.

Other approaches [2, 3] proposed enhancements for process models to express security requirements, but do not describe a schema to integrate these requirements in arbitrary modelling languages. Moreover, these approaches do not provide a meta-model that is capable to represent interacting participants in a Service-based system. To enable an automated generation of security policies for SOA, the roles and relation of these participants must be defined in the model as well.

In [4] Basin and Lodderstedt introduce SecureUML that provides a security design language to describe

role-based access control and authorisation constraints.

In summary, related approaches have been focused on the modelling of authorisation requirements or the specification of security requirements in the scope of business processes and do not support the generation of security policies. To support our model-driven approach, we present in this paper:

- An adaption of the schema introduced by Basin et al. [4] to define new security design languages for service-based systems.
- The abstract syntax and notion of our security modelling language SecureSOA used to express security intentions.
- The integration of SecureSOA in Fundamental Modelling Concept (FMC) Block Diagrams to enable the annotation of security requirements in system models.

Our security modelling language SecureSOA constitutes the foundation of our model-driven approach. Security intentions modelled in a language based on SecureSOA are translated to our meta-model for security policies in SOA that has been introduced in [5]. Finally, the security requirements defined in the meta-model are used to generate enforceable WS-SecurityPolicies.

The structure of this paper is as follows. In Section 2 we will introduce our concept to enhance design modelling languages with security intentions. Section 3 presents the syntax of our security design language SecureSOA to model these intentions. In the next section, a SecureSOA dialect based on FMC is introduced. Section 5 presents a use case that is modelled using this security design language. Our model-driven approach for SOA is outlined in the next Section. Section 7 presents related work, while Section 8 concludes the paper.

## 2 Modelling Security Intentions for SOA

Various modelling languages and dialects have been defined that can be used to model different aspects in an SOA. For instance, the system structure can be visualised in UML or FMC, while the processes executed by this system can be modelled with BPMN. Each modelling language provides a specific view on a particular aspect of the system that can be used to annotate security intentions. To aggregate and enforce intentions from different types of modelling languages, security intentions must be defined consistently and independent from any modelling language. In this sections we

discuss strategies to integrate security intentions into modelling languages and outline the steps of our approach.

### 2.1 Enhancing modelling languages

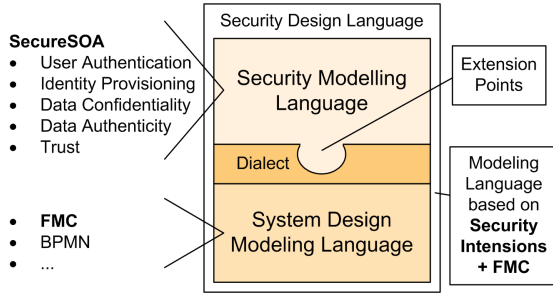
To integrate security intentions in system design languages, an enhancement of these languages is required. In general, three approaches can be distinguished to implement such an enhancement:

1. Light-weight extensions – The easiest way to enhance a particular system design language is the usage of extension points provided by the languages itself. For instance, UML provides stereotypes and tags to extend UML modelling elements. However, the visualisation of complicated security requirements might get confusing. Moreover, not all modelling languages define extension points to enhance modelling elements.
2. Heavy-weight extensions – Another approach to enhance modelling languages is based on the extension of its meta-model. For example, this approach is used by Rodríguez to define his security extensions for BPMN and UML [2] process models. The fact that the definition and integration of security requirements is done specifically for a particular system design modelling language based on its meta-model is one major disadvantage of this approach.
3. Defining a new language – To avoid the drawbacks mentioned above, a new modelling language can be defined. This modelling language integrates security elements and contain specific redefined elements of a system design modelling language. SecureUML uses this approach to model security requirements as an integral part of system models. Therefore, Basin and Lodderstedt described a generic approach to create a new security design languages by integrating security modeling languages into system design modelling languages as described in [4].

### 2.2 Defining Modelling Design Languages

The advantage of the schema described by Basin and Lodderstedt is its flexibility. A security modelling language can be defined once with certain extension points and can then be integrated into different design modelling languages for service-based systems. The resulting language is called a modelling dialect. Moreover, a

formal semantic can be defined for the security modelling language that enables the verification of the requirements modelled in any dialect. We have adopted this approach as shown in Figure 1.



**Figure 1. Schema for Constructing Security Design Languages**

The schema consists of the following parts:

1. A security modelling language is used to express security requirements for a specific purpose. We have defined SecureSOA that enables a modelling of security intentions for services-based systems.
2. The structure of a system is described using a system design modelling languages. While different types of modelling languages can be used, our approach is based on FMC Block diagrams that are used to visualise system architectures.
3. Both languages are integrated by merging their vocabulary using the extension points of the security modelling language. The resulting language is a called a dialect.

### 2.3 Merging Security and System Design languages

In SecureSOA, extension points are formalised entities that relate to security intentions. These extension points can be mapped to entities in any system design model.

For example, we formalise participants in an SOA such as services as objects that participate in an interaction by exchanging information. FMC visualises system architectures that are composed of agents communicating over a channel. Therefore, an object is an extension point and can be mapped to an agent.

As stated by Lodderstedt in [6], there is no universal approach to perform an integration of arbitrary security and design modelling languages. The integration technique depends on the structure of the se-

curity modelling language. In the scope of SecureSOA, we have identified three integration patterns that are listed in Table 1. The definition of these patterns is based on the classes and their relationships in the meta-models of the security and the system design languages. We denote the set of classes in the SecureSOA meta-model as  $s = \{s_1, \dots, s_{n_1}\}$ , classes in the meta-model of the security design language as  $m = \{m_1, \dots, m_{n_2}\}$  and classes in the meta-model of the dialect as  $d = \{d_1, \dots, d_{n_3}\}$ .

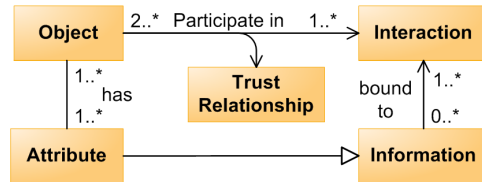
## 3 SecureSOA - a Security Design Language for SOA

SecureSOA enables a modelling of security intentions for service-based systems and is defined by a MOF-based meta-model (abstract syntax). In addition, we will introduce the notion of these elements (concrete Syntax) as UML profiles. Finally, the definition of the formal semantics of SecureSOA will be outlined in this Section.

### 3.1 SecureSOA Abstract Syntax


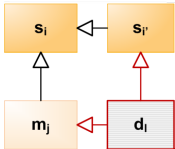
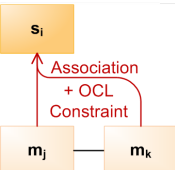
The SecureSOA meta-model [5] specifies the elements of this security language and consist of two parts. The meta-model for SOA introduces the basic entities in our model and their relationships to describe interactions in a Service-oriented Architecture. Based on this model, a model for security intentions and annotations is provided as well.

#### 3.1.1 A Meta-Model for SOA



**Figure 2. The Security Base Model**

As introduced in [7], one of the basic entities in our model is an *object* that consists of a set of *attributes*, participates in an *interaction*, and has *trust relationships*, see Figure 2. In addition, each interaction also involve the exchange of *information*. For instance in the scope of Web Services, an object could be a Web Service client or a Web Service itself. Therefore, we subclassify objects into *service*, *client* and *sts*.

<b>1) Subclass extension points</b>	
	<p>The easiest way to perform the integration is to map a class <math>m_j</math> in the design modelling language to its corresponding extension point <math>s_i</math> in SecureSOA that represents an abstraction of this class. This dependency can be easily represented as an inheritance relationship in the dialect between the classes <math>s_i</math> and <math>m_j</math>.</p>
<b>2) Enhance the dialect with new classes</b>	
	<p>In this case, a class <math>s_i</math> in SecureSOA can be mapped to a class <math>m_j</math> in the design modelling language as described by pattern 1. In addition, this class is inherited by a class <math>s_{i'}</math> of SecureSOA that models a specific aspect of service-based systems. However, it is unlikely to find a class in a general purpose modelling language that corresponds to this specialised class <math>s_{i'}</math>, although the abstract parent class <math>s_i</math> has been mapped. To associate the extension point <math>s_{i'}</math> with the design modelling language, it is necessary enhance the dialect with a new class <math>d_i</math> that inherits the class <math>s_{i'}</math> and <math>m_j</math>.</p>
<b>3) Define associations and OCL constraints</b>	
	<p>However, there might not be straight mapping for each extension point of SecureSOA, since certain aspects might be modelled on a different level of abstraction in both languages. In this case, a class <math>s_i</math> has to be mapped to multiple classes <math>m_j</math> and <math>m_k</math> in the other language. To integrate these classes into the dialect, association have to be defined between the corresponding classes. OCL constraints can be used to capture additional semantics of these dependencies.</p>

**Table 1. Schemas for Creating the Modelling Dialect**

To enable a detailed description of Web Service messaging, we model transferred information as *data transfer objects* as introduced by Fowler in [8]. In [5] we have shown the usage of this structure to describe SOAP messaging. A data transfer object has a *target* and an *issuer*. This reflects that a data transfer object can be send over several objects acting as intermediaries.

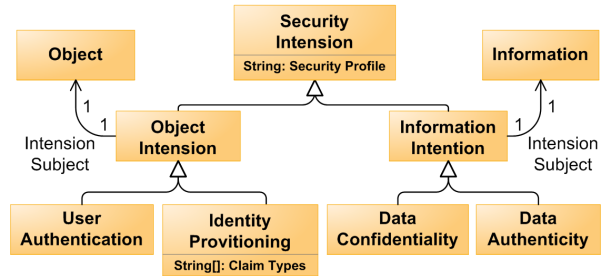
The classes defined in the scope of this SOA meta-model are the extensions points of SecureSOA as described in Section 2.2.

### 3.1.2 Modelling Security Intentions

Based on the SOA meta-model that has already been described in previous work [5], we have specified an enhancement of this model to express security intentions and annotations.

Security intentions are defined specifically for one security aspect in terms of service security and are related to one or more security goal. We have defined the following set of security intentions: User Authentication, Non-Repudiation, Identity Provisioning, Data Authenticity, Data Confidentiality, and Trust. Data Confidentiality, for instance, requires the security goal confidentiality for a particular piece of information, while identity provisioning states that the trustworthy iden-

tification and authentication of a user at a particular object is required. However, SecureSOA is not limited to this constraints and supports custom enhancements by adding additional security intentions.



**Figure 3. Modelling Security Intentions**

As shown in Figure 3, each security intention is related to a security profile. The fundamental idea is to hide technical details at the modelling layer. The modeller should not be bothered with details such as security algorithms and mechanisms that are used to enforce this intention. This set of information is predefined in a security profile and is then referenced by the security intention. Moreover, security intentions state requirements for a specific subject that is either an object or a data transfer object. Therefore, each security

intention has a intention subject. Object intentions refer to an object, while information intentions refer to an data transfer object.

While security intentions represent requirements, we use security annotations to represent security-related capabilities. For example, annotations could express identity claims that are supported by an identity provider. Annotations are defined similar to the security intention structure as shown in Figure 3.

### 3.2 SecureSOA Concrete Syntax

The concrete syntax specifies the visualisation of the element that are defined by the abstract syntax. Since the extension points of the SecureSOA meta-model (e.g. objects or interactions) are mapped to classes in the design modelling language, their notion is based on this language. Therefore, we just have to define the notion of security intentions and annotations.

In general, there are two possibilities to define a concrete syntax (notion) for these elements. The first option is to express them as a property of the subject of the element. Another option to visualise security requirements is the definition of artifacts for each element that can be used to annotate the element’s subject. We have chosen this approach to define the concrete syntax of security intentions specified by SecureSOA.

Our notion for security intentions is based on a UML concrete syntax using UML classes and stereotypes. Each intention is visualised as an UML class that is connected to the intention’s subject using an UML association. The mapping between SecureSOA intentions and UML stereotypes is listed in Table 2. The notion for security annotations is specified correspondingly.

UML Stereotype	Symbol
<< User Authentication >>	
<< Non-Repudiation >>	
<< Identity Provisioning >>	
<< Data Authenticity >>	
<< Data Confidentiality >>	
<< Trust >>	

**Table 2. SecureSOA Concrete Syntax**

### 3.3 SecureSOA Formal Semantic

We can formalize our SecureSOA meta-model as a relational model (based on sorts and relations) as described by Lodderstedt [6]. This formalisation facilitates the verification of the transformation in our

model-driven approach and will be outlined briefly: Classes and associations in the SecureSOA meta-model are mapped to a set  $c_i$  in a model  $m$  that contains an entry for each instance of a specific class or association. Each intention defines requirements concerning the exchange of information in a system  $m'$  that represents an secured instance of  $m$ . Therefore, the formal semantic of each security intention can be specified by defining an implication on the models  $m$  and  $m'$ . For example, if the security intention data authenticity relates to a data transfer object in the model  $m$ , then it will imply that this data transfer object must contain a signature in the model  $m'$ .

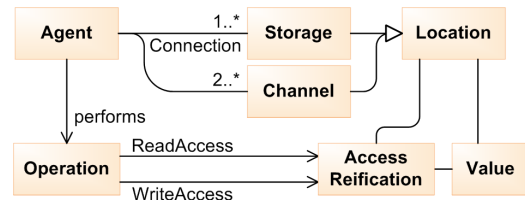
## 4 A SecureSOA dialect based on FMC

SecureSOA offers the possibility to express security intentions in various modelling languages. We have chosen FMC Compositional Structure Diagrams (Block Diagrams) as a system design modelling language, since FMC offers a suitable foundation to describe an SOA on a technical layer in terms of involved participants and their communication channels.

### 4.1 Fundamental Modeling Concepts

Fundamental Modeling Concepts (FMC) provides an approach to describe software systems. It can be used to model the structure of a system, processes in a system, and value domains of a system.

FMC Compositional Structure Diagrams (also known as FMC Block Diagrams) depict the static structure of a system and the relationships between system components. This diagram type distinguishes between active and passive components. Agents are active system components that are capable to communicate via channels and to perform activities in the system. Channels and storages are passive components used to transmit or store information.



**Figure 4. FMC Meta-Model**

The FMC meta-model [9] describes the abstract syntax for all diagram types and is specified using FMC entity relationship diagrams. We have translated the

FMC meta-model to a MOF-based meta-model. Figure 4 depicts the part of the meta-model that describes FMC block diagrams. Agents are connected to a storage or a channel that are locations and interact by performing read or write operations.

### 4.2 Merging SecureSOA and FMC

To integrate SecureSOA in FMC, the vocabularies of both languages have to be merged and the entities in FMC have to be mapped to corresponding extension points in SecureSOA. The meta-model of the dialect is shown in Figure 5.

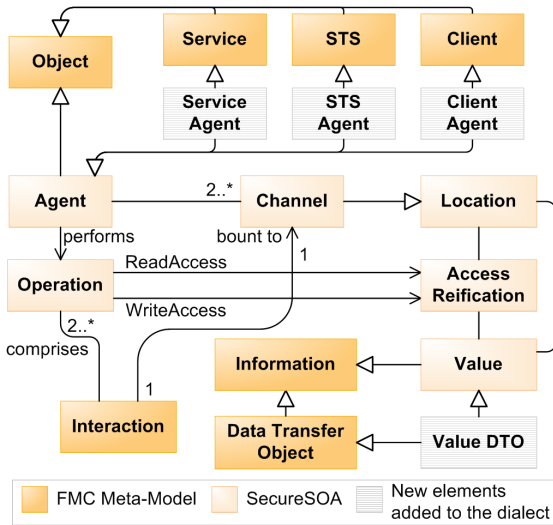


Figure 5. SecureSOA-based FMC dialect

As aforementioned in Section 2.3, the easiest way to perform the integration is to subclass elements of SecureSOA. Object is subclassed by Agent, while Information is subclassed by Value. However, there is no class in FMC that can be mapped to Service, Client, STS and Data Transfer Object. As described by our integration pattern 2 in Section 2.3, these extension points can be mapped by adding new elements (c.f. grey coloured elements in Figure 5) to the dialect that subclass related elements in FMC and SecureSOA.

Finally, the SecureSOA class Interaction has to be mapped to FMC. Subclassing will not work as integration technique, since interaction is not just a channel in FMC. It is composed of a channel in combination with an operation that is performed on this channel. Therefore, we defined associations and an OCL-Constraint to perform the integration as defined by pattern 3 in Section 2.3.

### 4.3 Defining the Concrete Syntax

The notion of the classes in the meta-model of the dialect is provided by the concrete syntax of FMC and SecureSOA. However, a notion must be defined for the elements Service Agent, STS Agent and Client Agent that have been added to the dialect. Since these elements inherit from FMC Agent, their notion is based on the notion this class. To indicate the agent’s type (Service, Client, or STS) we enhanced the concrete syntax with a notion for stereotypes as defined by UML.

## 5 SecureSOA Modelling Example

This section illustrates the usage of SecureSOA to model a web shop scenario as shown in Figure 6. This SOA scenario has been modelled using the web-based modelling tool Oryx [10]. We added our security design language introduced in the previous section as a stencil set to this tool.

The order scenario contains an order process, in which a user is requesting goods using an online store web application. This application invokes an composed order service that uses two external services; a payment and a shipping service. The payment service represents an external service which handles the payment of the order process. In order to do so, the service needs payment information including a payment amount and credit card information like card type, card holder, card number, expiration date and a security code. The shipping service initiates the shipping of the goods using the recipients address. Note that each agent indicates its type (Client, Service or STS) using stereotypes.

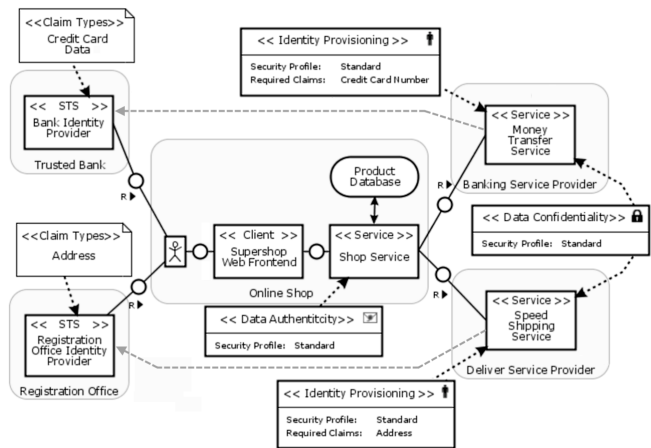


Figure 6. SecureSOA Web Shop Example

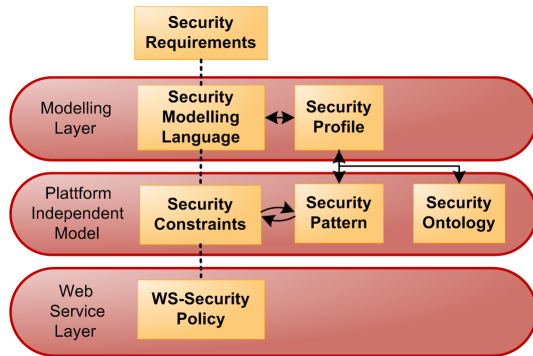
Users in this example have an account at their trusted bank and at the registration office, who act as

identity providers managing the user’s digital identity. The user can be authenticated at the identity providers to request a security token that can be used to access a specific service.

In addition, SecureSOA is used to annotate security intentions to various actors in this use case. The payment service has established a trust relationship with the trusted bank, while the shipping service trusts information from the registration office. Therefore, the money transfer service and the speed shipping service are annotated with the security intention identity provisioning, while the identity providers are annotated with the identity information that they offer. To secure the exchanged information, the intentions data authenticity and data confidentiality are used as well in this example.

## 6 Model-driven Security in SOA

SecureSOA enables the annotation of system design models e.g. FMC block diagrams or BPMN models with security intentions as shown in the previous Section. This language provides the foundation for our model-driven approach that enables SOA Architects to state security intentions at the modelling layer and facilitates a generation of enforceable security configurations [5]. As illustrated in Figure 7, our approach consist of three layers.



**Figure 7. Model-driven Security in SOA**

We use a policy meta-model to abstract from concrete policy languages (e.g. WS-SecurityPolicy) as introduced in [5]. A policy in this model consists of multiple security constraints that capture security requirements on a technical layer. Information at the modelling layer are gathered and translated to this model. To perform this transformation, expertise knowledge might be required to determine an appropriate strategy to secure services and resource, since multiple solutions might exist to satisfy a security goal. For ex-

ample, confidentiality can be implemented by securing a channel using SSL or by securing parts of transferred messages. Based on the security pattern approach [11], we have defined a formalised system of security configuration patterns that are used to resolve security constraints. Finally, these constraints are transformed into enforceable security policies based on a predefined mapping.

## 7 Related Work

The domain of model-driven security in the context of SOA and business processes is an emerging research area. Previous work done by Rodriguez et al. [12], [2] discusses an approach to express security requirements in the context of business processes. Although they support several security requirements, they neither describe a schema to integrate these requirements in other modelling languages nor describe a model-driven transformation.

Breu and Haffner proposed a methodology for security engineering in service-oriented Architectures [13] that is based on a model-driven approach. In particular, they outlined a transformation to authorisation constraints. Although providing a generic framework, they do not describe a mapping to WS-SecurityPolicy.

SecureUML [4] 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 approach presented in this paper.

Jürjens presented UMLSec [1] to express and verify security relevant information within UML-diagrams. However, the verification of security protocols and system models requires to express all security aspects at the modelling layer. This results in models that have a certain degree of complexity and do not provide a simple, high-level notion for security intention.

Wolter [3] fosters a model-driven approach to enable a generation of XACML access control policies based on enhanced business process models.

Jensen and Feja described a model-driven generation of Web Service security policies based on the modelling of security requirements in business process models [14].

Using security patterns, Delessy described a pattern-driven process for secure SOAs [15]. An automated translation to security policies is not described.

## 8 Conclusion and Future Work

System design modelling languages provide a suitable abstract perspective to specific security goals on a more accessible level. In this paper, we presented an approach to enhance arbitrary system design models with security intentions. Our approach is based on an universal schema that has been introduced by Lodderstedt and Basin in [4] to define security design languages. In this paper, we introduced SecureSOA as our security modelling language to express security intentions related to service security and described the concrete and abstract syntax of our language. Moreover, we discussed strategies to integrate our language into any system design modelling language. As a proof of concept, we defined a security design language by integrating SecureSOA in Fundamental Modeling Concepts Block Diagrams and added this language as a stencil set to the modelling tool Oryx [10].

To illustrate the expression of security intentions in FMC, we presented an order service scenario that is used to state security intention such as trust relationships, identity provisioning, and confidentiality. The specification of security intentions is the basis for our model-driven approach that addresses the difficulty to generate security configurations for Web Service systems. Altogether, our proposed modelling enhancement constitutes a suitable foundation to describe and implement a model-driven transformation of abstract security intentions to enforceable security configurations in different application domains.

In the next step, we will use the formal semantic of SecureSOA to verify the transformation process.

## References

- [1] Jan Juerjens. UMLsec: Extending UML for Secure Systems Development. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 412–425, 2002.
- [2] Alfonso Rodríguez, Eduardo Fernández-Medina, and Mario Piattini. A bpmn extension for the modeling of security requirements in business processes. *IEICE Transactions*, 90-D(4):745–752, 2007.
- [3] Christian Wolter and Andreas Schaad. Modeling of task-based authorization constraints in bpmn. In *BPM*, pages 64–79, 2007.
- [4] David Basin, Jürgen Doser, and Torsten Lodderstedt. Model driven security: from uml models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15(1):39–91, January 2006.
- [5] Michael Menzel and Christoph Meinel. A security meta-model for service-oriented architectures. In *Proc. SCC*, 2009.
- [6] Torsten Lodderstedt. *Model driven security: from UML models to access control architectures*. PhD thesis, Albert-Ludwig University of Freiburg, March 2004.
- [7] Christian Wolter, Michael Menzel, and Christoph Meinel. Modelling security goals in business processes. In *Proc. GI Modellierung 2008*, number ISBN 978-3-88579-221-5. GI LNI, Berlin, Germany, 1008.
- [8] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
- [9] Peter Tabeling, Rmy Apfelbacher, and Stefan Wappler. Fmc metamodel - the fundamental modeling concepts metamodel explained, September 2005.
- [10] Gero Decker, Hagen Overdick, and Mathias Weske. Oryx - an open modeling platform for the bpm community. In *BPM*, pages 382–385, 2008.
- [11] Joseph Yoder and Jeffrey Barcalow. Architectural patterns for enabling application security. In *PLoP*, 1997.
- [12] Alfonso Rodríguez, Eduardo Fernández-Medina, and Mario Piattini. Towards a uml 2.0 extension for the modeling of security requirements in business processes. In *TrustBus*, pages 51–61, 2006.
- [13] Michael Hafner and Ruth Breu. *Security Engineering for Service-oriented Architectures*. Springer, October 2008.
- [14] Meiko Jensen and Sven Feja. A security modeling approach for web-service-based business processes. *Engineering of Computer-Based Systems, IEEE International Conference on the*, 0:340–347, 2009.
- [15] Nelly A. Delessy. *A Pattern-driven Process for secure Service-oriented Applications*. PhD thesis, Florida Atlantic University, Boca Raton, Florida, May 2008.