A Pattern-driven Generation of Security Policies for Service-oriented Architectures

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

Robert Warschofsky
Hasso-Plattner-Institute
Prof.-Dr.-Helmert Str. 2-3
14482 Potsdam, Germany
robert.warschofsky
@hpi.uni-potsdam.de

Christoph Meinel
Hasso-Plattner-Institute
Prof.-Dr.-Helmert Str. 2-3
14482 Potsdam, Germany
meinel
@hpi.uni-potsdam.de

Abstract

Service-oriented Architectures support the provision, discovery, and usage of services in different application contexts. The Web Service specifications provide a technical foundation to implement this paradigm. Moreover, mechanisms are provided to face the new security challenges raised by SOA. To enable the seamless usage of services, security requirements can be expressed as security policies (e.g. WS-Policy and WS-SecurityPolicy) that enable the negotiation of these requirements between clients and services.

However, the codification of security policies is a difficult and error-prone task due to the complexity of the Web Service specifications. In this paper, we introduce our model-driven approach that facilitates the transformation of architecture models annotated with simple security intention to security policies. This transformation is driven by security configuration patterns that provide expert knowledge on Web Service security. Therefore, we will introduce a formalised pattern structure and a domain-specific language to specify these patterns.

1 Introduction

Service-oriented Architectures facilitate the flexible provision and reuse of services to enable a faster adoption to changing business requirements. One of the fundamental characteristics of SOA is the usage and orchestration of services in different application contexts. However, this flexibility comes along with new security risks and threats that require an individual protection of each orchestrated service. Sent and received messages must be protected and must convey identity information to enable the authentication and authorisation of users.

In general, these security requirements are stated in security policies and are provided with the interface description of the service. Service clients can retrieve the policy from the service and can use appropriate mechanisms to invoke the service securely.

In the scope of the Web Service specifications, WS-Policy and WS-SecurityPolicy provide an XML-syntax to state security requirements concerning the usage of WS-Security, WS-Trust and WS-SecureConversation. For instance, WS-SecurityPolicy can be used to specify requirements regarding the protection of exchanged messages (algorithms, key strength, protected message parts, ...) and the provision of identity information (Certificates, Username/Password, ...).

However, such policies are hard to understand and even harder to codify, due to the complexity of the Web Service specifications. To overcome these limitations, we foster a model-driven approach that generates security configurations based on system design models annotated with security requirements.

Modelling security has been a research topic in recent years. Some approaches (e.g. UMLsec [9]) enable the formal specification of security requirements in system diagrams, but tend to be difficult to understand without a strong security background. Other approaches [15, 19] proposed enhancements for process models to express security requirements on a more accessible level, but do not provide a mapping to security policy languages.

Our model-driven approach integrates security intentions in SOA system models using the integration schema introduced by SecureUML in [2] and enables a modeller to state basic requirements on a technically independent level. For instance, services modelled in system architecture diagrams can be annotated to require authentication.

The transformation of these intentions is challenging, since different strategies might exist to enforce a security intention. For example, confidentiality can be enforced at the transport layer or at the message layer and requires the provision of cryptographic keys.

The solution presented in this paper is driven by security configurations patterns for Web Services that represent security expert knowledge. To support our approach, we introduce in this paper:

- A transformation process that translates system models annotated with security intentions to security policies.
- The adaption and formalisation of the design pattern approach to represent security expert knowledge that guides the transformation process.
- A formalised structure and a domain specific language to specify security configuration patterns.

This paper is structured as follows. Section 2 introduces our model-driven approach and outlines the transformation process to security policies. The next Section provides background information about security patterns, while Section 4 introduces our formalised security pattern structure. The application of these patterns is described in Section 5. Section 6 describes related work, while Section 7 concludes this paper.

2 Model-driven Generation of Security Policies

Our model-driven approach simplifies the generation of security policies by enabling SOA Architects to state security intentions at the modelling layer and facilitats an automated generation of enforceable security configurations based on the modelled intentions. As illustrated in Figure 1, our approach consist of three layers. Security Requirements, expressed at the modelling layer are translated to a platform independent model. This model constitutes the foundation to generate WS-Security policies. The modelling of security requirements, the structure of the platform-independent model and the transformation process across these layers are explained in this Section.

2.1 Modelling Security Requirements

System design models such as FMC block diagrams or UML models are the foundation to enable system

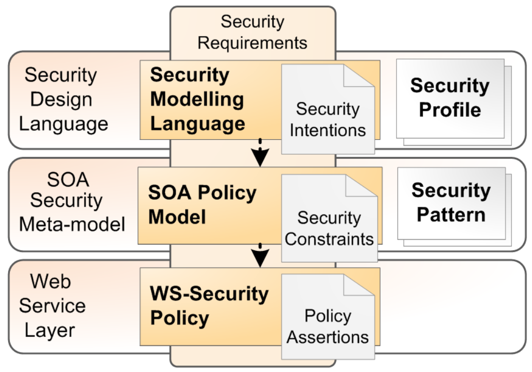

Figure 1. Model-driven Security in SOA

designers 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. SecureSOA uses the integration schema defined by SecureUML [2] to enable the enhancement of system design models with security-related modelling elements.

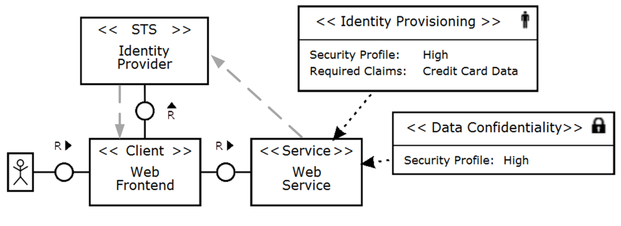

Figure 2. Modelling Security Intentions

A simple example diagram is shown in Figure 2. 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 tokens. These tokens can be send along with the request message to access the service.

In addition to the system structure, Figure 2 depicts two security intentions representing security goals that must be enforced by the security infrastructure.

Profile	Security Mechanisms
high	X509-Token
low	UserName-Token, X509-Token

Table 1. Security Profiles Examples

Each security intention refers to a security profile that is chosen by the modeller. In Figure 2, the profile *'high'* is used for both security intentions. Profiles are used to abstract from technical details that should be hidden from the modeller. For instance, instead of specifying the algorithms, key strength and other technical details, a modelled security intention refers to a profile that provide this information. Table 1 lists two profiles that are related to authentication.

2.2 A platform-independent model for security policies

The transformation from modelled security requirements to concrete security policies requires a platform-independent model that is capable to express these security requirements on an abstract layer, since security policy languages for SOA (such as WS-SecurityPolicy) just provide a syntax to state security requirements declaratively. In particular in the scope of WS-Security, it must be considered that

1. WS-Policy and WS-SecurityPolicy provide a syntax, but do not define a semantic. In fact, WS-Policy enables the negotiation and intersection of requirements between client and service without the need to know what is actually expressed by an policy option. However, to enable a mapping from security intentions to security policies, the meaning of the different requirements, their relation to security goals and their dependencies must be well known.

2. requirements that are semantically equal (for instance the encryption of message parts using different keys) have to be expressed in different ways in WS-Security-Policy.

Therefore, we use a policy meta-model that supports the expression of security requirements concerning communication related security goals as described in [11]. Our model serves as an abstraction layer for security policy languages, simplifies the handling of security policies, and enables the generation of security configurations in different policy languages.

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 Figure 3. 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.

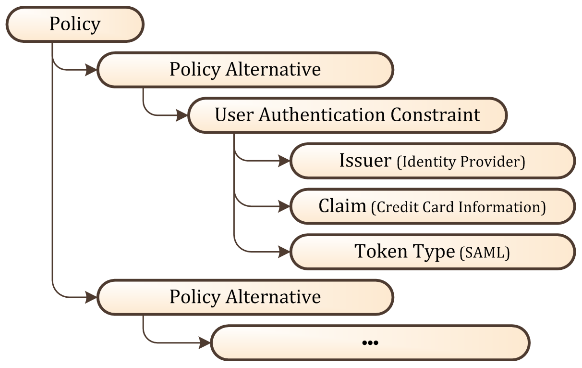
Policy
Policy Alternative
User Authentication Constraint
Issuer (Identity Provider)
Claim (Credit Card Information)
Token Type (SAML)
Policy Alternative
...

Figure 3. Policy Model Exmaple

2.3 A Pattern-based Transformation

Using the policy meta-model, the transformation of security intentions to security configurations works as follows: First of all, security constraints are generated based on the modelled security intentions and combined in policy alternatives.In a second step, these security constraints are transformed to a policy language.

The first transformation step from abstract security intentions to security constraints is quite challenging, since a simple mapping is not sufficient. Expertise knowledge might be required to determine an appropriate strategy to secure services and resource, since multiple solutions might exists to satisfy a security goal.

3 Security Pattern – State of the Art

As outlined in the previous section, expertise knowledge is required to determine an appropriate strategy that specifies how to secure a service by enforcing an intention.

In this section a short introduction to design patterns is given as well as an overview about the state of the art in security patterns that can be used to represent expert knowledge.

3.1 Design Patterns

The idea of design patterns has been introduced by Christopher Alexander in 1977: *'A pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that pattern'* [1]. This approach has been applied to software development in 1987 by Cunningham and Beck [3]. In general, design patterns are defined in an informal way, usually in the natural language, to enable programmer and system designer to adapt the solution described by a pattern to their own specific problem.Patterns are described in documents that have a specific structure as listed in Table 2. As described in [12], the mandatory elements of a pattern are *Name*, *Context*, *Forces*, *Problem*, and *Solution*.

Element	Description
Name	is a label that identifies the pattern and reflects the intention of this pattern.
Context	describes the environment before the application of this pattern.
Forces	are conditions that exist within the context.
Problem	describes a problem that occurs within the context.
Solution	is a proven solution for the problem within the context.

Table 2. Design Pattern Structure

3.2 Security Patterns for SOA Security

Security patterns have been introduced by Yoder and Barcalow [20] in 1997. Based on this work, various security patterns and pattern systems has been defined that refer to different phases in the development process. An overview about recent work is given in [21] by Yoshioka et al.

Delessy and Fernandez defined several security patterns for SOA and Web Service security[4, 6] that describe best practices and concepts such as *identity provider* and *identity federation*. These patterns provide an informal description, although parts of the pattern's solution are formalised using UML diagrams.

Microsoft published the book 'Web Service Security - Scenarios, Patterns, and Implementation Guidance' [18]. This book presents a catalogue of security patterns for Web Services and discusses the usage and preconditions for each pattern. In accordance with the design pattern structure, these pattern are described in an informal way.

The need of an formalization of security patterns has been addressed by M. Schumacher [17] by providing an classification. However, his approach is not suitable to enable an automated application of security patterns.

4 Formalizing and applying Web Service security pattern

Web Service Security patterns provide reusable expert knowledge that can be used by systems designers. As outlined in the previous Section, these patterns are represented in an informal way. In consequence, these approaches are not able to support a model-driven transformation based on an automated application of security patterns, since a formalisation of the pattern structure is required.

Therefore, our security pattern system is based on a formalised meta-model and a domain specific language that are explained in this Section.

4.1 The Data Model

As a foundation for our security pattern definition, we use our meta-model for SOA security that has been introduced in [11]. This model describes participants in an SOA – referred to as *Objects* – and the interaction between these objects by exchanging messages. In particular, we distinguish three types of participants: *Service*, *Client*, and *STS*. We can formalize this meta-model as a relational model (based on sorts and relations) as described by Lodderstedt [10]. Classes in the meta-model are mapped to a set that contains an entry for each instance of a specific class or association. For instance, the example shown in Figure 2 contains three participants (1: Web Frontend, 2: Web Service, 3: STS). These instances can be expressed as the following Sets: $Object = \{1, 2, 3\}$, $Client = \{1\}$, $Service = \{2\}$, and $STS = \{3\}$. In addition, the following relations can be defined that represent the interactions and trust relationships in the example: $OO_{Interaction} = \{(1, 2), (1, 3)\}$ and $OO_{Trust} = \{(2, 3), (3, 1)\}$

Security intentions (*Data Confidentiality, User Authentication, ...*) refer to participants as shown in Figure 2 and can specify multiple parameters such as required claim types. Similar to the formalization described above, security intentions can be represented as sets and association as well.

These sets and associations can be created based on the instances modelled in the system design model (see Section 2.1) and define the context for our security patterns.

4.2 Pattern structure

In our approach, a security pattern facilitates the generation of security constraints (as introduced in Section 2.2) for a specific security intention. The applicability of a pattern depend on the forces of this pattern that specifies conditions in the scope of the context provided by the system design model.

Figure 4 illustrates relationship between our security configuration patterns and our data model. The structure of our security patterns is defined as follows:

name – A String that identifies the pattern.

problem – The problem addressed by a security configuration pattern is identified by a security intention.

context – As mentioned in the previous section, the context is determined by the entities and their relationship described in the system model.

forces – Forces determine the applicability of a pattern in a specific context and can be expressed as conditions over the entities and their relations in the context.

solution – The solution to the problem defines operations that results in the instantiation and configuration of security constraints that has been described in Section 2.2. These constraints comply with the security intention referenced by the problem. Moreover, the solution might require the usage of other patterns that results in the generation of additional security constraints.

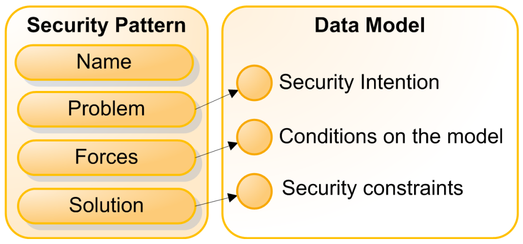

Figure 4. Security Configuration Pattern

To sum up, our security patterns provide a solution in terms of security constraints for a security intention. The applicability of a pattern is determined by the context and related forces.

4.3 A domain Specific Language for Security Configuration Patterns

To specify the forces and the solution of a pattern, a language is required that provides a syntax to state conditions and operations on the data model. Languages such as QVT [14] or ATL [13] have been specified in the scope of the Model-Driven Architectures (MDA) approach and provide an expressive and standardized syntax to define model transformations. However, the flexibility of these languages comes along with an increased complexity that would complicate the definition of a security pattern. Therefore, we propagate the usage of a concise domain specific language (DSL) that is defined specifically for the usage in our security pattern system.

A security configuration pattern DSL operates on a data model that is represented by domain-specific sets and relations. The transformation of system design models to such a representation has been introduced in Section 4.1. Consequently, our DSL operates on three basic data types that we refer to as *values* with ($value = set \,|\, number \,|\, boolean$). A set can contain multiple elements of type value.

We use the following sets to represent different types of actors: *'Service'*, *'Client'*, and *'STS'*. Moreover, relations might be defined between these participants that are represented by the following functions in our DSL:

1. $Interaction(<number>, <number>)$ and $Trust(<number>, <number>)$ determine whether there is a trust or interaction relationship between two objects and return a boolean value.

2. $InteractionPath(<number>, <number>)$ and $TrustPath(<number>, <number>)$ determine whether two objects relate concerning the transitive closure of the respective relations and return a set of numbers that represent the objects on the path.

A security pattern provides a solution for a problem that is identified by a security intention. As outlined in Section 4.1, a security intention refers to a specific subject and multiple parameters. The concrete instance of an intention that is referenced by an applied pattern is denoted as *intention*. The related parameters and the subject of this intention can be accessed as properties. For instance *intention.subject* returns a number that identifies the participant.

So far, we have defined basic sets and functions related to the domain-specific data model. Next, we will introduce mathematical functions that provide the foundation to express the condition of the forces in a pattern. Therefore, each of the following operations return a boolean value:

<set> CONTAINS <value> – checks whether an value is contained in a set. If this value is a set, then it is verified that each element of the second set is contained within the first set.

<value> AND/OR <value> – these operators correspond to the boolean operators. If the expression is a set, then it is considered whether the set has elements or not.

<value> IMPLIES <value> – represents an implication.

Moreover, we have to define *Pattern-specific operations* that support the expression of a pattern's forces and solution.

FORALL (<set>){ <operation> }* – executes a list of operations for each element in a given set.

ENSURE (<value>) – is used to define the forces of a pattern. A pattern can be applied in a certain context, if the value evaluates to true.

ENFORCE (<security_intention>) – this operation can be used in the scope of a solution and indicates that a specific security intention must be enforced. A set of security constraints is returned that is created by the application of appropriate patterns for the intention.

Finally, we have to specify *domain-specific operations* that facilitates the creation and manipulation of security constraints.

REQUIRE (<String>) – this operation results in the creation of security constraints that are returned by the solution.

SET (<String>, <Object>) – sets a specific property (identified by a String) of the constraints that has been created during the execution of the solution.

USE (<String>) – similar to the *SET-operation*, this operation sets a specific property at the constraints. The value is resolved from the profile as introduced in Section 2.1.

Using our DSL, we can define the forces and the solution of a pattern as a list of operations: $pattern.forces = operations*$ $pattern.solution = operations*$.

4.4 A Security Pattern Example

Name:	Brokered Authentication
Problem:	User Authentication

Forces:

```
ENSURE ((intention.subject IN STS)
      OR (intention.subject IN Service))
FORALL (Clients) {
      ENSURE (InteractionPath(it, intention.subject) IMPLIES
                  TrustPath(intention.subject, it)) }
```

Solution:

```
FORALL (TrustPath(intention.subject, Clients)) {
      ENFORCE('Authentication') }
```

Table 3. Pattern ’Brokered Authentication’

As an example, we will introduce the formalisation of the brokered authentication pattern for Web Services as described in [18]. As shown in Table 3, the forces of this pattern state that the subject of the intention must be an STS or a Service. Moreover, for each client it must be ensured that an interaction with the subject implies a trust relationship between the subject and this client. This means that an interaction between a client and a service requires a trust relationship to enable the subject to authenticate the user. For example, consider the application of this pattern to the service in Example 2. The forces are fullfiled, since the subject is a service and the client that interacts with this service, has a trust relationship with this service (established over the STS).

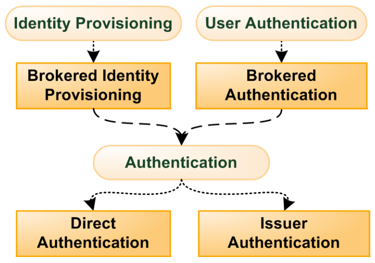

Figure 5. Authentication Patterns

The solution of this pattern states, that the security intention ’Authentication’ must be enforced for each element on the trust path. In Example 2, the intention ’authentication’ will be enforced for the Service and the STS. This will result in the application of additional security patterns (see Section 4.3). Figure 5 shows a part of our security configuration pattern system for Web Services. The security patterns *’Direct Authentication’* and *’Issuer Authentication’* will generate security constraints for the participants in the trust path to ensure that the STS authenticates the user and that the service authenticates the credential issued by the STS.

5 Applying Security Patterns

The application of security patterns is performed by a pattern engine. This engine applies a security pattern system to the security intentions modelled in the system design model and returns a set of resolved security constraints (see Section 2.2) that can be transformed to a policy language. An important feature of a pattern engine is the capability to interpret and enforce the operations of the DSL that are specified in the previous section.

As aforementioned, a security pattern operates on a data model that is described by a system design model. Therefore, the translation of the model to sets and relations is the first step as described in Section 4.1. These information are stored in a data structure called *execution context*.

The pattern engine provides a method *'resolve intentions'* that expects the execution context and a list of security intentions as input parameters and returns a set of security constraints. For each modelled security intention, the pattern engine has to find security patterns that refer to the required intention and whose forces evaluate to true as illustrated in figure 6.

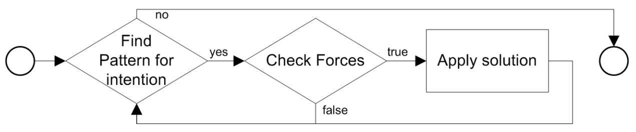

Figure 6. Pattern Application Process

The application of the solutions of the applied patterns result in the instantiation of security constraints that are created by the operations specified in Section 4.2. If a solution contains an *ENFORCE*-operation, then a recursive execution of the process illustrated in Figure 6 will be required. The security constraints generated by the application of additional security patterns are added to the set of new security constraints.

For each security intention, multiple patterns might be applied. The sets of security constraints created by the application of these solutions represent policy alternatives. For instance, the enforcement of the security intention *secrecy* might result in the application of two patterns and, therefore, the creation of two constraints: one constraint that requires security at the transport layer and one constraint that requires security at the message layer.

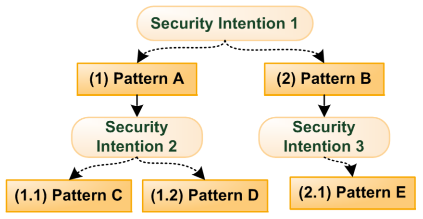

Figure 7. Pattern Application Tree

Moreover, since each solution might require other security intentions, the enforcement of an security intention can be visualised as a tree, as shown in Figure 7. Therefore, the generation of policy alternatives works as follows: All constraints that are on the path from the root element to a leaf in this tree are combined in a policy alternative. The policy alternatives are expressed in our model and can be transformed to a concrete security policy language.

To proof the applicability of our approach, we implemented a pattern engine based on groovy. For this purpose, we used groovy's capabilities to specify domain specific languages.

6. Related Work

Modelling security configurations in Service-oriented Architectures is an emerging topic.

Rodríguez *et al.* [15] and Wolter [19] proposed enhancements for process models to express security requirements. However, these diagrams do not provide a suitable foundation to model communication-related security requirements that can be transformed to executable policies.

This has been addressed by Breu and Haffner [7] who proposed a methodology for security engineering in service-oriented Architectures.In particular, they outlined a transformation to authorisation constraints. Although providing a generic framework, they do not considers specific Web Service characteristics such as claim-based identities and do not describe a mapping to WS-SecurityPolicy.

Jensen and Feja described a model-driven generation of Web Service security policies based on the modelling of security requirements in business process models [8]. In particular, their approach intends to generate policies that ensure a secure messaging in terms of confidentiality and integrity.

SecureUML [2] 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 [9] 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, high-level notion for security intentions.

Satoh and Yamaguchi introduce an intermediate model to transform a WS-SecurityPolicy into platform-specific configuration files for WS-Security [16]. This model is defined to represent the WS-Security message structure and the meanings of signatures and encryption specified in a WS-SecurityPolicy. However, our policy meta-model provides more flexibility compared to the intermediate model of this approach.

In the recent years, various security patterns have been defined. A detailed discussion about related work concerning security patterns in the scope of SOA security has been presented in Section 3. Using these security patterns, Delessy described a pattern-driven process for secure SOAs [5]. An automated translation to security policies is not described.

7. Conclusion and Future Work

Specifications for security policies in SOA such as WS-Policy and WS-SecurityPolicy provide a language to enable a declarative configuration of security requirements. Due to the complexity of these languages, the creation of security policies is an error-prone task.

To simplify the management and creation of security policies, we presented an model-driven process in this paper that enables the generation of security configurations based on modelled security intentions. This transformation process is driven by Web Service security configuration patterns that represent reusable expert knowledge. Since security patterns are tradtionally stated in an informal way, we introduced a formalised pattern structure in this paper to enable an automated application of security patterns.

The foundation of our security configuration pattern is our SOA meta-model [11] that describes entities and their relations at the modelling layer. The forces of our security configuration pattern state conditions on the entities in this model that describe the applicability of a pattern. A security configuration pattern provides a solution that results in the creation of security constraints. These constraints are described by our policy meta-model that serves as an abstraction layer to WS-SecurityPolicy and enables the transformation. To enable the specification of the forces and solutions in a simple and understandable way, we introduced a domain-specific language that provides operations that are specific for our approach.

In the next step, we will extend our policy generation approach to create aggregated security policies in the scope of service compositions.

References

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

[2] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: from uml models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15(1):39–91, January 2006.

[3] K. Beck and W. Cunningham. Using pattern languages for 0bject-oriented programs. Technical Report CR-87-43, AppleComputer, Tektronix, September 1987.

[4] N. Delessy, E. B. Fernandez, and M. M. Larrondo-Petrie. A pattern language for identity management. In *ICCGI '07: Proceedings of the International Multi-Conference on Computing in the Global Information Technology*, page 31, Washington, DC, USA, 2007. IEEE Computer Society.

[5] N. A. Delessy. *A Pattern-driven Process for secure Service-oriented Applications.* PhD thesis, Florida Atlantic University, Boca Raton, Florida, May 2008.

[6] N. E.B.Fernandez and M. Larrondo-Petrie. Patterns for web services security. In *OOPSLA : Workshop on Service-Oriented Architecture and Web Services*, 2006.

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

[8] 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*, 0:340–347, 2009.

[9] J. 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.

[10] T. Lodderstedt. *Model driven security: from UML models to access control architectures.* PhD thesis, Albert-Ludwig University of Freiberg, March 2004.

[11] M. Menzel and C. Meinel. A security meta-model for service-oriented architectures. In *Proc. SCC*, 2009.

[12] G. Meszaros and J. Doble. A pattern language for pattern writing, 1996.

[13] I. OBEO. Atlas transformation language 3.0. Specification, June 2009.

[14] OMG. Meta object facility (mof) 2.0 query/view/ transformation specification. OMG Specification, April 2008.

[15] A. Rodríguez, E. Fernández-Medina, and M. Piattini. A bpmn extension for the modeling of security requirements in business processes. *IEICE Transactions*, 90-D(4):745–752, 2007.

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

[17] M. Schumacher. *Security Engineering with Patterns - Origins, Theoretical Model, and New Applications.* Number ISBN 3-540-40731-6. Springer, Berlin, 2003.

[18] A. Stamos and S. Stender. *Web Service Security.* Microsoft Press, 2005.

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

[20] J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In *PLoP*, 1997.

[21] N. Yoshioka, H. Washizaki, and K. Maruyama. A survey on security patterns. *Progress in Informatics*, 5:35–47, 2008.