SECTISSIMO: A Platform-independent Framework for Security Services

Mukhtiar Memon, Michael Hafner, and Ruth Breu

University of Innsbruck, AUSTRIA
{mukhtiar.memon, m.hafner, ruth.breu}@uibk.ac.at

Abstract. It is non-trivial to secure dynamically composed systems based on language- and technology-independent service components. One of the approaches to tackle the challenge is the use of powerful *Security Modeling Frameworks* based on tools to generate security executables. We propose the SECTISSIMO framework: a layered approach for the modeling of security-critical, service-oriented systems. In SECTISSIMO the functional models are enriched with security extensions and transformed into executables using *Abstract Security Protocols* and *Controls*. Based on them, we generate *Security Policies* containing Authentication and Authorization Assertions to execute on target platform. On the target platform the *Security Components*, provide security functionality to enforce the generated policies. The components are integrated based on the principles of the Service Component Architecture (SCA) and provide interfaces to access the their functionality.

1 Introduction

Inter-organizational workflows among business partners involve sharing of resources owned by partners (i.e. service providers) and consumers alike. Shared resources may contain personal data of individuals as well as legal and financial information of the enterprises. The workflows processing security-sensitive data need to be modeled explicitly with security and privacy considerations. The common practice has been to meet the security-related requirements in the systems during implementation with language-dependent handcrafted fixed-code. Such inflexible code can not withstand the unforeseen challenges emerging from changes in business logics, workflow variations and patchy platform technologies. A better approach is to define security requirements in functional models and write transformation rules rather than writing security programs. This is the spirit of model-based engineering and also the conceptual foundation of the SECTISSIMO framework. We investigate security critical use cases in service-oriented systems and apply appropriate security patterns and protocols to meet their security requirements followed by model-transformation to generate executable code. In particular, with the SECTET framework [14] we have developed an approach for the systematic design and realization of security-critical workflows for web services technology. It consists of UML-based elicitation of security requirements, a *Reference Architecture* based on web services and a prototypical *Code Generator*. SECTISSIMO takes the crucial step beyond in providing a platform that abstracts from the underlying technology and forms the security services from abstract security artifacts .

The remainder of the paper is organized as follows. In *Sec.*2 we discuss related approaches. We present the proposed SECTISSIMO framework concepts, details of its layers and example healthcare scenarios in *Sec.*3. This is followed by discussion of *Implementation Specific Model* in *Sec.*4, *Security Components* in *Sec.*4.1 and Service Component Architectures in *Sec.*4.2 respectively. For implementation, the required *Security Technologies* are discussed in *Sec.*4.3. We conclude with discussion of future work in *Sec.*5.

2 Related work

Several approaches deal with the modeling of security requirements using security patterns and protocols. In [21] security goals are modeled for cross-organizational business processes, which considers SOA-based federated environment that comprises multiple independent trust domains. Single Sign-on based Authentication is solved with security modeling in [19], which also separates the application layer from the security layer. Other approaches which focus on security patterns for modeling security requirements are extensively covered in [12]. Some work within the Security Engineering community deals with specification of security requirements in the context of formal methods. Examples are the UML extension UMLsec [15] together with the AUTOFOCUS tool and the PCL approach [11]. Few groups deal with aspects of code generation in the context of secure solutions. Among these are the groups of Basin [17] and Ulrich Lang [16]. Both approaches present frameworks for high-level specification of access rights including code generation; the former in J2EE environment and the latter in CORBA. Our method exhibits some similarities, but has the advantage that we introduce an additional level of abstraction between models and code. Moreover, our approach goes further in that we do not only deal with access rights but also with other security requirements such as *Non-repudiation, Rights Delegation, Privacy* and *Auditing*. Among other approaches are important theoretical results like the RBAC model [13], access models in AKENTI [20] and PERMIS [10]. We rely on some ideas of these approaches like the role or credential concepts, however, we have a rather wider domain in our focus. The technological foundation of SECTISSIMO framework is conformable to Service Oriented Security Architecture (SOSA), as we compose security services from the components, which offer security functionality. These components are integrated based on *Service Component Architecture* (SCA) model [18].

3 SECTISSIMO: A Platform-independent Framework for Security Services

The SECTISSIMO framework allows to model security requirements in parallel with business processes. It is a three-layered approach comprising; 1. *A Secure Business Process Model*, 2. *An Abstract Security Services Model*, and 3. *An Execution Platform* Layers. SECTISSIMO is a novel concept, as most of the approaches in security modeling consider two layers (i.e. *Security Modeling* and *Transformation*). Common approaches generate security assets for specific platform, where the transformation rules are bound to the target platform. Hence, in case of the changes in the security requirements or target platform, new transformation rules have to be written to generate artifacts. To overcome this limitation, the proposed framework introduces an additional layer of abstraction between the security-enhanced models and the implementation technologies. In this abstraction layer, security requirements are modeled with more fine-grained details using *Security Protocols*, *Security Controls* and *Composition Rules*. As the security-enhanced functional models contain abstract security requirements, which need more details of the target platform before they can be transformed into security code (e.g. Executable Security Policies). The layer of abstraction between models and transformation provides space for modeling such details. This layer uses abstract security artifacts such as security protocols and maps them to concrete security functionality to be executed at the target platform.

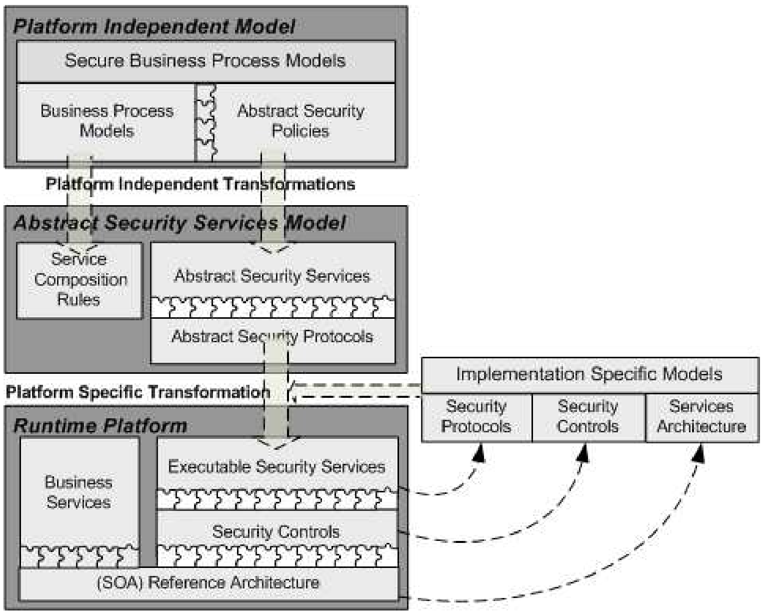

Platform Independent Model
Secure Business Process Models
Business Process Models
Abstract Security Policies
Platform Independent Transformations
Abstract Security Services Model
Service Composition Rules
Abstract Security Services
Abstract Security Protocols
Platform Specific Transformation
Implementation Specific Models
Security Protocols
Security Controls
Services Architecture
Runtime Platform
Business Services
Executable Security Services
Security Controls
(SOA) Reference Architecture

Fig. 1. SECTISSIMO Framework

Figure 1 sketches the three-layered architecture of the SECTISSIMO framework, conceived as a substantial extension to the two-layered approach in our previous SECTET framework [14]. The *Secure Business Process Model* provides the functional view of the system enhanced with security concerns. The *Abstract Security Services Models* integrate a repository of *Platform-independent Security Artifacts* based on Security Protocols (e.g. Authorization, Single Sign-on, Non-repudiation) and Security Controls (e.g. Algorithms for Encryption/Decryption, Digital Signature). The set of *Service Composition Rules* specify the order in which the security protocol executes in a given scenario and the conditions that have to be fulfilled before protocol composition. For example, confidential document exchange is executed in the order i.e. *Signature* → *Encryption* at the sender's end and *Decryption* → *Signature Verification* at the receiver's end.

3.1 Example of Secure Healthcare Service Scenarios

In the SECTISSIMO framework, we differentiate between two domains: the *Security* domain and the *Application* domain. Both domains contain meta-concepts defined in their respective meta-models. For example, the *Security Domain Metamodel* (please refer to [9] for comprehensive information) defines meta-concepts of security such as *Roles* and *Documents*. Whereas, a *Healthcare Metamodel* is an example for an Application domain, which uses healthcare related meta-concepts such as *PatientRecord*, *EmergencyAccess* and *4-Eyes Principle*. With this separation of domains, the application developer can focus on functional requirements and leave the security problems to be solved to the security expert [19].

To explain the framework mechanisms, we present an example in the following subsections. It addresses use cases for securing patient's medical data. Different users in the healthcare domain (i.e., Physician, Patient, Pharmacy, Insurance etc.) with different permissions (which depend upon their legal rights and job responsibilities) store, process and share the medical data [22]. We will be discussing following two scenarios related to security of patient's data:

> SCENARIO 1: *Identity Resolution: when a user accesses services accross different Security Domains, the Single Sign-on Security Pattern may be used to realize this requirement.*
> SCENARIO 2: *Security Policies for Authorization to let legitimate users access the services. This will be modeled with the Authorization Pattern.*

In the first phase, we model the *Secure Business Process Models* as shown in *Figure* 2. We use the *Security Domain Metamodel* from [9], which defines the security concepts and their relations more comprehensively. *Figure* 2 shows an example healthcare *Application Domain* model, with security enhancements stereotyped using meta-concepts from the *Security Domain Metamodel*. Using these models, we discuss abovementioned scenarios and working mechanism of the proposed framework and explain the responsibilities of its different layers.

3.2 Business Process Layer

The first layer comprises functional models enhanced by security requirements, so called *Security Policies*. *Figure* 2 shows the security-enhanced model for (a part of) a healthcare system as a UML class diagram. It consists of the main entities `Physician` and `PatientRecord`. The class diagram is extended based on the *UML Profile Extension Mechanism* through *stereotypes* [7].
The stereotypes i.e. `<<Role>>` and `<<Document>>` are the meta-classes of *Security Domain Metamodel*. The OCL constraint associated to the `PatientRecord` class defines an `Authorization Constraint` i.e. the physician can access the data only within timings of 9:00 to 17:00 hours. The other role shown in the role hierarchy is an `Specialist`, who has limited access to patient's data depending upon her expertise. The second OCL constraint defines an `Authorization Constraint` for the `Specialist` i.e. if her *Specialization* is *Heart Specialist*, only then she can access the patient's data related to *Heart Surgery*. The medical record is composed of the *Diagnosis* report and a *Prescription*.

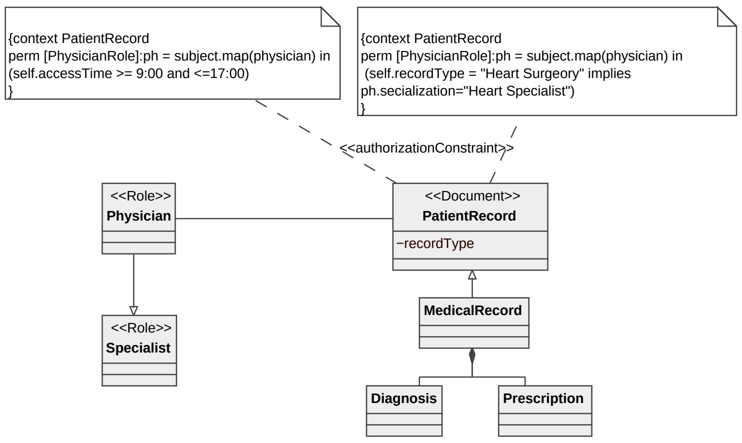

{context PatientRecord
perm [PhysicianRole]:ph = subject.map(physician) in
(self.accessTime >= 9:00 and <=17:00)
}
{context PatientRecord
perm [PhysicianRole]:ph = subject.map(physician) in
(self.recordType = "Heart Surgeory" implies
ph.secialization="Heart Specialist")
}
<<authorizationConstraint>>
<<Role>>
Physician
<<Document>>
PatientRecord
−recordType
MedicalRecord
<<Role>>
Specialist
Diagnosis
Prescription

Fig. 2. Healthcare System Model with Security Policies

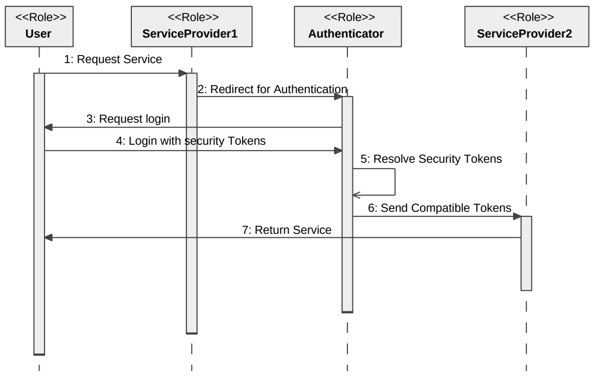
3.3 Abstract Security Services Layer

The second layer provides an additional abstraction for the security services which execute on target platform. In order to provide this abstraction, security requirements defined in models are mapped to the security patterns and protocols which can solve those requirements. This is referred as *Platform Independent Transformation* in *Figure* 1. In the example scenarios, the two security requirements i.e. Authentication and Authorization, are mapped to the security patterns of *Single Sign-on* and *Authorization* respectively. Subsequently, these patterns are elaborated with the security protocols to show detailed interaction among protocol participants. The Security Protocols and Controls required to model the example scenarios are discussed in following subsection.

3.4 Abstract Security Protocols and Controls

The *Security Protocols*, which can solve a variety of security problems are the protocols for Authentication, Authorization, Single Sign-on, Non-repudiation and Privacy. The Security Protocols use *Abstract Security Controls* mapped to concrete security controls at the target platform. The mapping of abstract to concrete security controls is done by the *Implementation Specific Model* as shown in *Figure* 1. The ISM integrates this mapping to the *Platform Specific Transformation*, so that when transformation rules are written, those should take into consideration the compatibility of the type (or format) of the artifacts to be generated. The concrete security controls are the security programs for encryption/decryption, digital signature and token/assertion generation performed while sending/receiving messages between protocol participants and security services for Authentication, Authorization and non-repudiation etc. These programs are written in specific *Security Components*, which offer security functionality to implement them (*Section* 4.1).

In SCENARIO 1, in order to solve the security problem of *Identity Federation* (When a physician accesses health services from other security domain), the *Single Sign-On*



<<Role>> User
<<Role>> ServiceProvider1
<<Role>> Authenticator
<<Role>> ServiceProvider2
1: Request Service
2: Redirect for Authentication
3: Request login
4: Login with security Tokens
5: Resolve Security Tokens
6: Send Compatible Tokens
7: Return Service

Fig. 3. Single Sign-on Authentication

(SSO) Authentication pattern is used (*Figure* 3). The *User*, *Service Providers* and *Authenticator* are stereotyped as `<<Role>>` . The credentials of the user are validated by `ServiceProvider1`, who has a trust relation with `ServiceProvier2`. The `Authenticator` validates user's credentials for `ServiceProvider1`, but those credentials are not compatible with the security domain of `ServiceProvider2`. The `ServiceProvider2` mentions in its Security Policy that he accepts SAML assertions from `Authenticator` before sending the requested service. An example WS-SecurityPolicy of `ServiceProvider2` is shown below [8].

```
<wsp:Policy>
   <sp:SamlToken sp:IncludeToken="AlwaysToRecpt">
       <wsp:Policy>
            <sp:WssSamlV11Token10/>
       </wsp:Policy>
   </sp:SamlToken>
</wsp:Policy>
```

The `Authenticator`, which in this case is a `SAML Authority` creates a SAML response for the user and sends it to the user. The user adds the SAML response to the Security Header of SOAP message for service request and sends the request to the `ServiceProvider2`, who returns the service. An example of the SAML response assertion carrying user's authentication statement is shown below, where a user i.e. Dr. Peter with `<<Role>>` *Physician* is authenticated as a subject.

```
<saml:Assertion
         xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
         ...
         AssertionID="xbhutysxmigruieLoPiuex98xkioU"
         Issuer="Authenticator"
         IssueInstant="2008-06-22T11:05:41.795Z">
    <saml:Conditions NotBefore="2008-06-22T11:00:41.795Z"
        NotOnOrAfter="2008-08-22T15:41:22.795Z"/>
    <saml:AuthenticationStatement
     AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
     AuthenticationInstant="2008-06-22T11:41:20.608Z">
        <saml:Subject>
            <saml:NameIdentifier>Dr. Peter</saml:NameIdentifier>
        ...
            </saml:Subject>
    </saml:AuthenticationStatement>
</saml:Assertion>
```

In SCENARIO 2, the physician accesses the services offered by *ServiceProvider*. She gets access based on her access rights which depend upon her job responsibilities [13]. The *Authorization Security Pattern* solves the problem of modeling the authorization policy for the physician. As shown in *Figure* 4, the Authorization Service checks the user rights and constraints and sends *permit* or *deny* decisions in the *AuthzDecision-Statement* part of SAML Authorization Assertion (The code for authorization assertion is not provided due to space limitation).

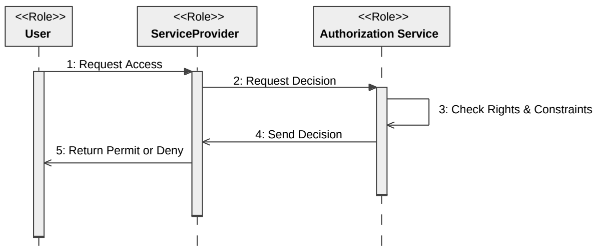


Fig. 4. Security Protocol for Authorization

3.5 Service Composition Rules

The *Abstract Security Protocols* use *Service Composition Rules*, which decide the criteria on the basis of which the protocols are composed. Modeling security protocols needs knowledge of the participants (i.e. `ServiceProvider1, ServiceProvider2` and `Authenticator`), pre-conditions required to be fulfilled, or post-obligations to be agreed upon by the participants. These rules perform following checks for protocol composition:

1. *Selection of Protocol Participants*: These rules select the particular business partners which collaborate in the business process. When this rule is applied for *Single Sign-on Authentication*, it selects the participants (i.e. `ServiceProvider1` and `ServiceProvider2`) and *Authenticator* as shown in *Figure* 3.
2. *Pre-Conditions*: Before the `ServiceProvider1` and `ServicePorvider2` collaborate, they must agree upon the terms and conditions involved in collaboration. These terms and conditions may involve the legal or ethical requirements to be agreed upon beforehand. Also, if the collaborators want to negotiate certain service parameters (e.g. monitory), could be resolved using pre-conditions.
3. *Post-obligations*: Rules defining post-obligations to be taken into account while (or after) executing protocol (e.g. payments, auditing).

The composition rules can be written in formal way with protocol composition logic. In models, rules can be defined with Object Constraint Language (OCL), which provides constructs for setting conditions on model elements such as classes, instances and attributes (detail beyond the scope of this paper).

3.6 Execution Platform Layer

The *Execution Platform Layer* is where the secure business workflows execute. It is based on a runtime platform targeting an specific software application or middleware technology such as web services or CORBA. The runtime platform consists of a Reference Architecture which relies on security mechanisms and primitives as technology-dependent counterparts of the Abstract Security Protocols and Controls. The security artifacts, which execute at runtime include Security Policies containing Authentication and Authorization Assertions. In the example scenarios, the security artifacts are *Authorization Policies* defining access rights for a physician and *Identity Assertions* when she accesses services from different security domains.

4 Implementation Specific Model

After modeling security protocols the next phase in the framework is the *Platform Specific Transformation* of *Abstract Security Protocols* into the security services, which execute on target platform. Before writing transformation rules, it is necessary to get the platform details on which business services execute. This includes information related to the *Service Architecture*, the *Security Protocols* and the *Security Controls* of the target platform. The details are expressed in an *Implementation Specific Model* (ISM), which integrates them into *Transformation* functions to generate compatible security artifacts. The ISM plays a role of an *Adapter* between models and code to harmonize the execution of security artifacts at the target application. The business services executing on a target platform belong to a variety of application formats such as J2EE/.NET applications, Web Services, BPEL Processes, CORBA components etc. These applications require different formats of security artifacts such as WS-SecurityPolicy, J2EE Application Deployment Descriptor and CORBA's Policy Definition Language (PDL).

4.1 Security Components

The target platform needs to execute various security services. For example, the implementation of requirement of Single Sign-on necessitates a security infrastructure that accepts security tokens (i.e. username/password) from `ServiceProvider1` and create a SAML Assertions for `ServiceProvider2`. A Security Components is the piece of implementable code that executes such functions related to security controls such as encryption, digital signature, token/assertion generation etc. It uses the security infrastructure to create tokens (i.e. username/password, Kerberos ticket), generate keys and certificates etc. The component are written in the languages such as Java, C++ and can be called with its interfaces.
The use of the component paradigm, on one hand enhances the flexibility in the selection of most appropriate component and on the other hand it lets the application developer focus on the system functionality and leave the security-related implementation to be met by the security expert. For an implementation that uses two or more different component services a *Security Service Composite* is formed and executed. The composition of components is performed by *Component Composition Service* based on the principles of *Service Component Architecture* (SCA) paradigm.

4.2 The Service Component Architecture (SCA)

The *Service Component Architecture* (SCA) paradigm, provides component-based model featuring *Loose-coupling, Flexibility, Composition* and *Productivity* [5]. An SCA component has two types of interfaces; *Provided* and *Required* (also called *Service References*) interfaces. They are used to integrate with other components to form the service

composite. The composition of components occurs through the *wiring* of service interfaces and references. The essential elements of SCA, which provide the architectural flexibility to form composition of components and network of services are *Development* (Development of individual components), *Composition* (Composition of components into services) and *Assembly* (Structure of composite services or service networks).
Figure 5 shows the composition of security components for Authentication, Authoriza-

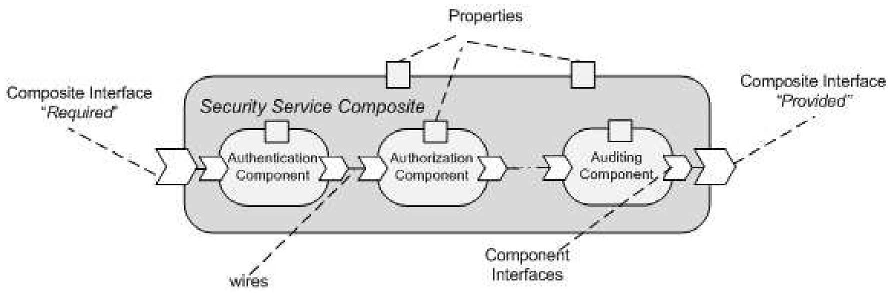


Fig. 5. Service Component Architecture for Security Components to form a Composite

tion and Auditing. When integrated these components form a *Security Service Composite*. The ISM defines which security component is suitable to be implemented for which security protcol. For example, the *Authentication* component implements the protocol for *Single Sign-on* and *Authorization* component implements the protocol for *Authorization*. For SCA implementation, recently Apache-Tuscany provides an API to write the service composites from the components written in Java and C++ [1]. An *SCA-Policy Framework* is provided by OpenSOA [5] and OASIS-OpenCSA [3]. The framework provides security services to components using *Security Intents*. Our approach of providing security is different from the one implemented by openSOA's SCA-Policy Framework. Because, the components referred in these specifications provide application functionality, whereas, we use the components that provides security functionality. Also SCA-Policy Framework considers implementation of the security intents as WS-Policy only. On the contrary, we consider the implementations that are best suitable to the target application and runtime platform (i.e. WS-SecurityPolicy, J2EE Deployment Descriptor etc.). Security composites can be written in XML-based Service Composition Definition Language (SCDL) [6].

4.3 Transforming Models into Security Technologies

In our framework, we export UML models into XML Metadata Interchange (XMI). XMI enables easy interchange of metadata between UML-based modeling tools and MOF-based metadata repositories in distributed platform-independent environments [2]. For writing transformation rules we use openArchitectureWare(oAW)[4], which provides a family of languages for model-to-model and model-to-code transformation. The security APIs used for writing security components are based on Java, XML and WS-security. For *Portable Identity* in Single Sign-on, we use Security Assertions Markup Language (SAML) for Authorization Policies we use eXtensible Access Control Markup Language (XACML). We use WS-SecurityPolicy to define security requirements attached to particular service.

5 Conclusion and Future work

In this paper, we proposed the SECTISSIMO framework, which is a three-layered approach that transforms security-aware high-level functional models into the artifacts which execute on a target platform. We assume that any services (e.g. Web Services, BPEL Process) or applications (e.g. Web, J2EE/.NET) can execute these services which use the code generated from abstract security artifacts. Our present work in SECTISSIMO is an enhancement to the SECTET framework [14] and in future plan to add more security capabilities for deperimeterized SOA applications.

References

1. Apache-Tuscany Project. http://incubator.apache.org/tuscany/.
2. MOF 2.0 / XMI Mapping Specification. http://www.omg.org.
3. OASIS - Open Composite Service Architecture. http://www.oasis-opencsa.org.
4. OpenArchitectureWare(oAW). www.openarchitectureware.org.
5. Service Component Architecture. http://www.osoa.org/display/Main/Home.
6. Service Component Definition Language. http://www.davidchappell.com.
7. UML Profile Extension Mechanism. http://www.omg.org.
8. WS-SecurityPolicy. www.oasis-open.org.
9. M. Alam, M. Hafner, M. Memon, and P. Hung. Modeling and Enforcing Advanced Access Control Policies in Healthcare Systems with SECTET. In *MOTHIS '07: MODELS 2007*, Nashville, USA, 2007.
10. D. W. Chadwick and A. Otenko. The PERMIS X.509 role based privilege management infrastructure. In *SACMAT '02: Proceedings of the 7th ACM symposium on Access control models and technologies*, pages 135–140, New York, NY, USA, 2002. ACM.
11. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *J. Comput. Secur.*, 13(3):423–482, 2005.
12. R. David, G. Carlos, F. Eduardo, and P. Mario. Security patterns and requirements for internet-based applications. *Internet Research*, 16(5):519–536, 2006.
13. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.
14. M. Hafner, R. Breu M. Breu, B. Agreiter, and Andrea Nowak. SECTET: An Extensible Framework for the Realization of Secure Inter-Organizational Workflows. *Journal of Internet Research*, 16(5), 2006.
15. J. Juerjens. *Secure Systems Development with UML*. SpringerVerlag, 2004.
16. U. Lang and R. Schreiner. *Developing Secure Distributed Systems with CORBA*. Artech House, Inc., Norwood, MA, USA, 2002.
17. T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 426–441, London, UK, 2002. Springer-Verlag.
18. G. Peterson. Service Oriented Security Architecture, 2005. http://www.arctecgroup.net.
19. F. Satoh, Y. Nakamura, and K. Ono. Adding Authentication to Model Driven Security. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, pages 585–594, Washington, DC, USA, 2006. IEEE Computer Society.
20. M.R. Thompson, A. Essiari, and S. Mudumbai. Certificate-based authorization policy in a PKI environment. *ACM Trans. Inf. Syst. Secur.*, 6(4):566–588, 2003.
21. C. Wolter, M. Menzel, and C. Meinel. Modelling security goals in business processes. In *Modellierung*, pages 197–212, 2008.
22. G. Yee, L. Korba, and R. Song. Ensuring Privacy for E-Health Services. In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security*, pages 321–328, Washington, DC, USA, 2006. IEEE Computer Society.