

Model-Driven Security Based on a Web Services Security Architecture

Yuichi Nakamura, Michiaki Tatsubori, Takeshi Imamura, and Koichi Ono

IBM Tokyo Research Laboratory

nakamury@jp.ibm.com tazbori@jp.ibm.com imamu@jp.ibm.com onono@jp.ibm.com

Abstract

The emergence of Web services and Service-Oriented Architecture (SOA) makes application development easy. However, since the computing environments on which applications are running are becoming complex, it is harder for users to set up security properly. Considering such complex security environments, this paper describes a tooling framework to generate Web services security configurations using Model Driven Architecture (MDA). According to the MDA concept, users simply add security intentions to an application model, and then detailed security configurations are generated, employing transformations over UML constructs and a security environment model. In order to demonstrate that the framework is practically useful, we also illustrate how to generate configuration files for a commercial product.

1. Introduction

The emergence of Web services and the Service Oriented Architecture (SOA) [1] makes application development easy because application components called *Web services* can be coupled over intranets and via the Internet. Meanwhile, the configuration of non-functional aspects such as security and quality of services is becoming difficult. The underlying computing environments on which applications are running are becoming complex because computers can be networked in complicated topologies, including firewalls and intermediate servers. Setting up the non-functional aspects requires a fairly deep understanding of such complex environments.

Regarding the security that we address in this paper, we also have to precisely understand the security infrastructure. According to the Web services concept, services located in various businesses are connected. Since each business has its own security infrastructure, integration of such infrastructures must be considered. The Web services security (WS-Security) roadmap [2] proposes a framework for security federation in which we can integrate multiple security domains, even with different security mechanisms such as X.509 [3] and Ker-

beros [4]. Although the framework enables security federation in principle, it requires a user's deep knowledge of security, and an understanding of the security infrastructures.

We believe that security must be unified with the software engineering process, and thus *security engineering* [5] is important. Unfortunately, security is considered as an afterthought in most actual development in the sense that security is added after the functional requirements are implemented. It is well known that finding defects downstream greatly increases the costs of removal and repair.

Although there are some challenges from a security engineering perspective as presented in [5], we address a post-deployment issue. It is still important to help programmers implement secure code. However, in the SOA context, it is more important to consider how to configure security at the deployment. Since existing applications are coupled over various network topologies, adding security code to applications is not realistic. Thus, security is typically configured when distributed applications are integrated.

In previous work [6], we have already shown a tooling framework for generating WS-SecurityPolicy [7] from business-level security intent, leveraging a collection of transformations. In this paper, we enhance the previous tooling framework in two senses. First, we better align our work with actual development processes employing the Model Driven Architecture [8]. In particular, taking an SOA perspective, we can cover security artifacts at both the business and technology levels in an orderly fashion. Second, we demonstrate how to extend our framework for a commercial application server product, using a complicated scenario. As described in Section 4, we provide a detailed platform model, taking account of the WS-Security architecture, security infrastructure, and the architectural design of the application server product. Configuration files for the product are generated, using the transformations and the platform model.

Our ultimate goal is to explicitly represent all security artifacts including the business- and technology-level features, and the relationships among them. In this way,

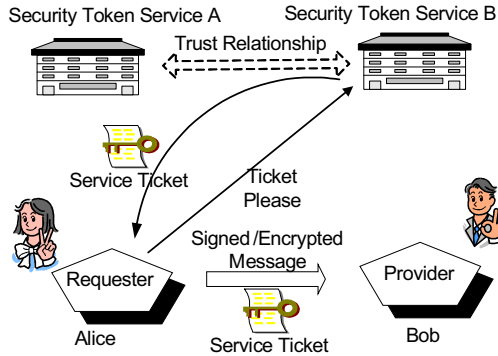


Figure 1. Federation Sample in WS-Security

even business users can ensure that running applications meet their business-level security intentions. Furthermore, security compliance can be validated over the security representation.

The rest of this paper is organized as follows: Section 2 introduces a WS-Security architecture, and discusses its configuration issues. In Section 3, we describe the architecture of our tooling framework. SOA is extended in terms of security, and key constructs of tool are described. A detailed example is shown in Section 4 in order to demonstrate our idea on top of a commercial product. Section 5 discusses related work, and some remaining issues in our work. In Section 6, we conclude this paper.

2. Configuration Issues of Web Services Security

2.1 Web Services Security

WS-Security is intended for a framework to federate multiple security domains beyond the security mechanisms for Web services. Because applications are integrated across business boundaries according to the Web services concept, Web services have specialized security requirements. There already exist numerous security domains on the Internet and in intranets. Although applications and business entities belong to one or multiple more security domains, they have to interact with one another even beyond the security domain boundary. Thus, the federation of different security domains is extremely important, although the issue hasn't been addressed in existing security technologies, especially between different security mechanisms.

The WS-Security architecture discussed in the roadmap document [2] addresses the issue of integration of security domains. Figure 1 illustrates a simple federation between two security domains. The requester, Alice, accesses a Web services provided by Bob while they belong to different security domains. The Security Token Service (STS) provides a ticket called a security token to access a Web service. When both requester and provider belong to the same security domain, they will

access their STS. Even when they belong to different domains, they can securely communicate via a federation mechanism. As shown in Figure 1, when there is a trust relationship between two STSs, Alice under STS-A can get a ticket from STS-B so as to access Bob's service. This kind of federation can work even if the security mechanisms are different, i.e. Kerberos vs. X.509.

In order to support the federation scenario, the Web Services Security core specification (WSS) [9] is quite extensible and flexible. Specifically, users can define their own security tokens that can be used to digitally sign or encrypt any parts of the messages. While such extensibility serves as a basis for security, it is often hard to set up security configurations properly.

2.2 Tooling Issues

We have developed commercial WS-Security tools for the IBM WebSphere Application Server [10] and Rational Application Developer [11], enabling users to process any kind of WS-Security messages and to add arbitrary security tokens. While we provide a complete configuration function, there are more than seventy parameters that must be properly set by users. Obviously, such numerous parameters cause usability issues. More importantly, users can easily create security holes.

In order to overcome the usability issues, we here envision how users think about security. User thinking can be outlined as follows:

1. Users identify participants such as business entities, and typical interactions among them.
2. Users evaluate risks and threats, and identify security requirements at a higher abstraction level
3. Users refine the abstract security requirements into detailed policies, considering various factors such as their company's security infrastructures.
4. Users set up detailed security parameters for the given runtime systems such as application servers and database systems.

Current commercial products seem to provide tools only for Step 4. However, from a usability perspective, it is useful to support the other steps because users can begin by considering abstract security intent.

3. Architecture of Model Driven Security

Security is often considered only during implementation or deployment phases. Considering security as afterthought is not a good idea. Our thesis is that we should take account of security from the beginning of application development.

In order to organize a security-tooling framework, we leverage two ideas: Model-Driven Architecture (MDA) and Service-Oriented Architecture (SOA). MDA is an approach to application design and implementation. Even at the early stages of development, abstract system models are constructed, and they are further refined to the next concrete level. The initial abstract model is

gradually elaborated toward a sufficiently detailed model that is converted to program code. We want to refine the abstract security intent along with the MDA elaboration processes.

In contrast, SOA provides a perspective to define abstraction levels, as we will further describe in Section 3.1. The levels spread out from business objectives to implementation details. In other words, SOA provides a framework to refine business goals into implementations.

In our approach, we define security primitives at each abstraction level defined by SOA, and elaborate the security primitives of one level into the next concrete levels. In this way, we can associate the security requirements of all of the levels so that we can maintain them even with future changes.

Here, we first describe SOA in more detail, then give an overview of our security tool. In our tool, users can attach security annotations to application designs represented in UML [12]. The security annotations are detailed toward the implementation level using transformation patterns. During the transformations, we also take account of the security environment where the application is deployed.

3.1. Service-Oriented Architecture and Security

SOA emphasizes the importance of business aspects such as revenue, cost, and reputation and better integration with the IT-level (Information Technology) artifacts. Business decisions are made by a business executive such as a Chief Executive Officer (CEO). Therefore, business objectives concerning a CEO are considered in SOA, and they are refined to come up with the IT-level details in application programs.

We define the following four levels for discussion:

- **Strategy Level** – Business objectives involve an organization’s strategic goals, business design, and key performance indicators for revenue and profit.
- **Operation Level** – Business processes are modeled, including organizational structure and resources such as people and utility services.
- **Execution Level** – The business processes are described in an executable manner. Here we assume specific execution engines such as J2EE and Business Process Execution Language (BPEL) [14].
- **Deployment Level** – Applications developed at the Execution level are still independent of the target platforms. When deploying, we have to specify many parameters that depend on the platform including hardware, system software, network infrastructure, and middleware.

We apply the MDA concepts to security. We can use UML at the lower three levels, though it is not clear how to model with UML at the strategy level. Therefore, we here begin by addressing the operation level in order to

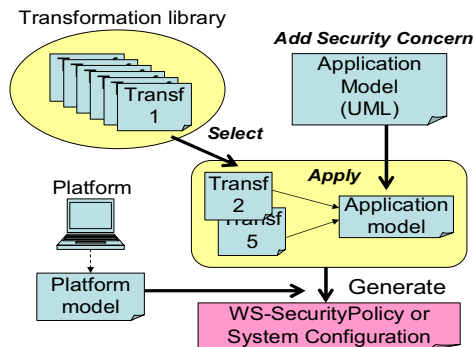


Figure 2. Tool Overview

include security. We summarize what is described for security at each level as follows:

- **Operation Level** – Only security intentions are added to the business process models. Primitives for security intentions should be abstract enough so that business users can understand them.
- **Execution Level** – Security requirements derived from the intentions are added to the executable artifacts such as the J2EE application packages. We assume that security requirements are described in the configuration of the application packages, i.e. the deployment descriptor (DD) for J2EE.
- **Deployment Level** – The security requirements at the Execution Level are bound to specific security infrastructures where applications are running. In other words, we describe *how* to fulfill the requirements at this level.

We believe that even a business user like a CEO has to understand the security aspects of their applications to make business decisions. While the goal of this paper is to generate a security configuration, we want to leverage the security artifacts at all levels to assure the CEO that the security intentions are properly reflected in the running applications.

3.2. Model Driven Security Tool

Security intentions at the operation level are transformed towards the deployment level in our tool, as illustrated in Figure 2. Application Model indicates an abstract application design represented in UML. Users first add security intentions to the model with abstract security primitives. A security intention is elaborated by applying a Transformation that maps security primitives to particular elements in a target configuration.

In our tooling framework, we can take arbitrary descriptions for the target description. While we used WS-SecurityPolicy in our previous paper [6], we target configuration files for a commercial product in this paper. Specifically, we generate DD and binding for IBM WebSphere Application Server (WAS), taking account of its architecture design as described in Section 4.

Table 1. Examples of Security Primitives

Primitive	Intent
Authentication	Used to denote a party that has to be authenticated in the scope of a given operation
Integrity	Used to denote that the data transferred between parties must be guaranteed to reach the recipient in the same form and with the same content
Non-repudiation	Used to denote that the marked information includes the digital signatures of the parties related to the document
Confidentiality	Used to denote that the marked information should be treated as private (protected against unauthorized viewing)

During the generation, we have to take account of the details of the security infrastructure where the application is deployed, because the target description requires technical details such as algorithm names and key lengths. The Platform Model represents the information of such security environments.

Here, we describe the key constructs of the tool: primitives for security intentions, transformations, and the platform model. A detailed example appears in Section 4, taking a complicated scenario with WAS.

3.2.1. Security Primitives for UML

Although there has been a lot of research on security issues concerning technologies, we want to address business-level security intent that is easy to understand even for business users. Here, we adapt the security primitives proposed in [14] because they are abstract enough and their representation in UML is also discussed. Table 1 shows examples of the primitives. Since we intend to use them for WS-Security, we have picked some of primitives in [14], but the names were changed to WS-Security terminology.

The primitives in the table are represented as UML stereotypes. Therefore, they can be added to UML diagrams without violating the UML syntax. Since we assume that business users look at the UML diagrams to understand the business processes, they can also see the security intentions in the same diagrams.

3.2.2. Transformations

A transformation is a rule that refines an abstract security intention into elements in a detailed description. Sources and targets of transformations are both represented in UML, as in Figure 3. For the source, the service invocation is represented in UML classes and security primitives such as integrity are added to the service class. The transformation target here is configuration information for WAS. Although the configuration in WAS is described in XML, we can define an object model for the configuration which is represented in UML as in the figure.

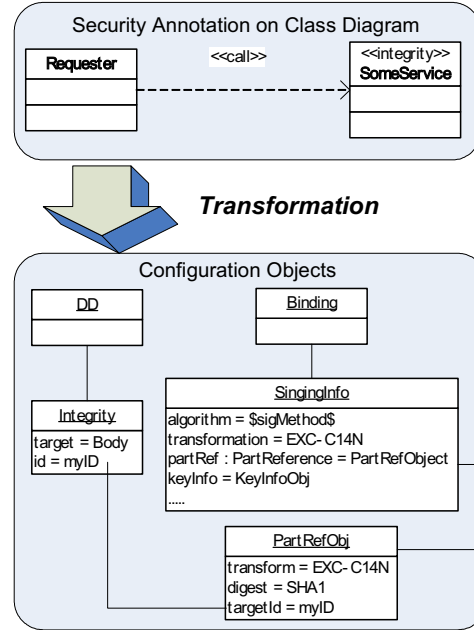


Figure 3. Transformation Sample for Integrity

Let us take a closer look at the example. WAS configuration consists of two files: DD and binding. As discussed in Section 3.1, the DD is an artifact at the Execution level while binding is at the Deployment level. In the DD, the Integrity object defines that the body of the messages must be targeted for the integrity check. In contrast, in the binding, the SigningInfo object specifies a signing algorithm, a transformation method, and a reference to a part of DD. Note that the algorithm property of the SigningInfo object is a variable. The property value is determined by referring to a platform model described in Section 3.2.3. Furthermore, PartRefObject specifies which Integrity object in the DD is associated with it, providing more detailed information such as how to canonicalize (C14N) the signed part, and how to calculate the digest value for the canonicalized part.

In practice, we need to collect a complete set of transformations. We have already identified a collection of transformations, targeting WS-SecurityPolicy as described in [6] and [15]. In our approach, we defined *idioms* as building blocks to characterize message protection. Combining the idioms, we can describe transformations in an orderly fashion.¹

¹ In our previous papers, transformations are called best practice security patterns. Since patterns are UML constructs, we use transformations in this paper to avoid confusion.

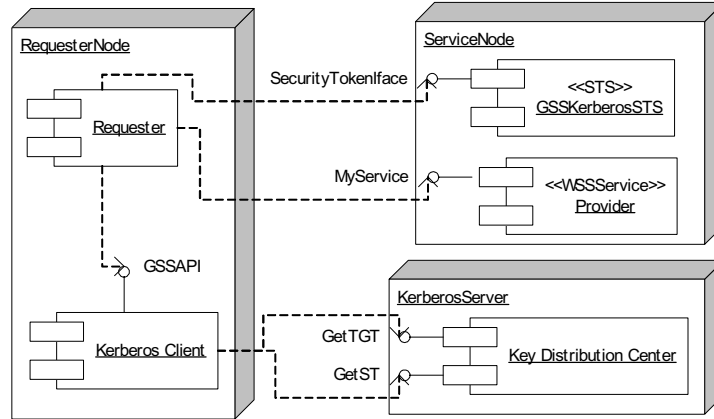


Figure 4. Example Platform Model

3.2.3. Platform Model

In order to determine the detailed parameters for a security configuration, we have to represent the security environment where the applications are deployed. A platform model is an abstraction of the security environment that pays particular attention to WS-Security features. The platform model in our initial prototype had only limited features because it was only a proof of concept for our transformation-based security tooling.

We extend the previous platform model considering the following items:

- UML Representation: While the previous tool was written in XML to capture only the minimum set of features, we adopted UML to represent various aspects of the security environments.
- System Details: Security configuration often requires knowledge of the runtime architecture of the target systems, such as the application servers. Some features of the target systems are explicitly represented in the new model.

An example of a platform model is illustrated in Figure 4. This model shows a runtime environment for WS-SecurityKerberos [16] which is described in more detail in Section 4. The model contains three UML nodes: RequesterNode, ServiceNode, and KerberosServer. Each node has some UML components such as Requester and Provider. The processing here is carried out as follows:

1. Requester invokes KerberosClient using the Generic Security Service (GSS) API [17].
2. KerberosClient interacts with the Key Distribution Center (KDC) to get the Kerberos tickets.
3. Requester invokes GSSKerberosSTS with the obtained Kerberos ticket to get a security token.
4. Requester invokes Service with the obtained token, signing and encrypting the request messages.

The detailed properties are included in the model though they are not shown in the diagram. For example,

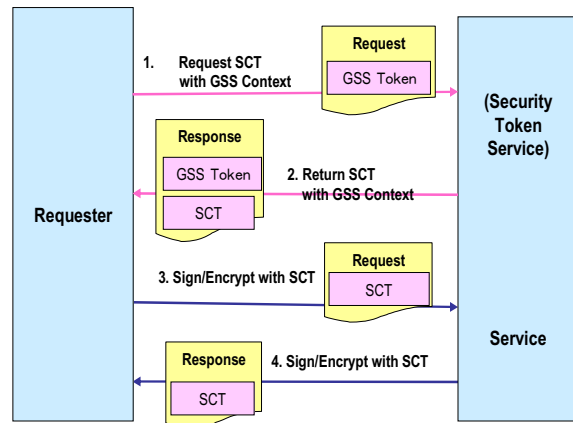


Figure 5. WS-SecurityKerberos Overview

GSSKerberosSTS has the type of security token returned by this STS, and Provider has a required security token type and an acceptable signature method.

4. Detailed Example

Here, we present a scenario in more detail, using a WS-SecurityKerberos (WS-SK) implementation² developed on top of WAS.

Let us take a look at an overview of WS-SK as shown in Figure 5 (with reference to Figure 4, also). WS-SK is performed as follows:

1. Requester sends a GSS token which wraps a Kerberos service ticket

² Technology preview is found at

<http://www-306.ibm.com/software/webservers/appserv/was/>

```

<S:Envelope xmlns:S=".../soap-envelope">
  <S:Header>
    <wsse:Security xmlns:wsse=".../secext">
      <wsse:SecurityContextToken wsu:Id="sct"
        xmlns:wsu=".../utility">
        <wsu:Identifier>som-Id</wsu:Identifier>
      </wsse:SecurityContextToken>
      <ds:Signature xmlns:ds=".../xmldsig#">
        <ds:SignedInfo>
          ...
          <ds:SignatureMethod
            Algorithm="...#GSS_MIC"/>
          <ds:Reference URI="#body">...</ds:Reference>
        </ds:SignedInfo>
        ...
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#sct"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </S:Header>
  <S:Body wsu:Id="body">...</S:Body>
</S:Envelope>

```

Listing 1. WS-SecurityKerberos Message

2. STS returns a special security token, called, Security Context Token (SCT), along with an updated GSS token. With the GSS token, the requester can share a *secret key* with a service.
3. Requester sends a request message to the service, signing the message with the shared key.
4. Service sends a response message, signing it with the shared key.

It must be noted that the interaction with KerberosServer shown in Figure 4 does not appear in the WS-SK protocol. However, when initiating the conversation, Requester needs to know the detailed KDC information that is captured by the platform model.

A sample message with WS-SK is shown in Listing 1. The SOAP Body element is signed with a GSS-specific signature method, called, GSS-MIC, and the SecurityContextToken element is used to indicate a signing key.

The WS-SK message is generated with a pair of configuration files: The Deployment Descriptor (DD) and the binding. The DD in specifies that the body part of the message requires integrity, and thus that part will be

```

<clientServiceConfig>
  <securityRequestGeneratorServiceConfig>
    <integrity>
      <messageParts target ="body"/>
    </integrity>
  </securityRequestGeneratorServiceConfig>
  <securityResponseConsumerServiceConfig />
</clientServiceConfig>

```

Listing 2. Deployment Descriptor Sample

signed (see bold part).

On the other hand, the binding file in defines how to achieve the integrity specified in the DD. As shown in the listing, its contents are pretty complicated, and therefore it is often hard to specify all parameters properly by hand.

In order to understand the binding file format and semantics, let us take a look at the architecture of the WS-Security handler in WebSphere as shown in Figure 6. In the figure, Requester is a requestor application that creates a request message, and WSSGenerator is an entity that adds a WS-Security header entry to the request message. Since WS-Security requires extensibility for some portions such as the signing algorithms and security tokens, several plug-in points are provided around WSSGenerator, such as SigEngine and TokenGenerator.

Walking through the architecture, WS-SK is taken as an example. The processing is performed as follows:

1. Requester invokes WSSGenerator
2. WSSGenerator invokes TokenGenerator to create a SecurityContextToken (SCT)
3. Since no SCT is cached initially, TokenGenerator invokes JAAS CallbackHandler
4. CallbackHandler interacts with STS to get a GSS Token and an SCT
5. TokenGenerator stores a GSS token associated with an identifier of the SCT
6. WSSGenerator invokes SigEngine, giving the SCT identifier
7. During the signing, SigEngine eventually needs a key indicated by the SCT identifier, so KeyLocator is invoked.
8. KeyLocator finds a key with the identifier
9. SigEngine signs the message, using the provided key

```

<securityRequestGeneratorBindingConfig>
  <signingInfo>
    <signatureMethod algorithm="GSS_MIC"/>
    <canonicalizationMethod algorithm="exc-c14n"/>
    <partReference part="int_body">
      <transform algorithm="exc-c14n"/>
      <digestMethod algorithm="sha1"/>
    </partReference>
    <signingKeyInfo/>
  </signingInfo>
  <keyInfo>
    <keyLocatorMapping locatorRef="gen_klocator"/>
    <tokenReference tokenRef="gen_sigtgen"/>
  </keyInfo>
  <keyLocator name="gen_klocator"
    classname="KerbKeyLocator"/>
  <tokenGenerator name="gen_sigtgen"
    classname="SCTGenerator">
    <valueType localName="SCT"/>
    <callbackHandler
      classname="SCTCallbackHandler"/>
  </tokenGenerator>
</securityRequestGeneratorBindingConfig>

```

Listing 3. Binding Sample

With this architecture in our mind, it is easier to understand the binding sample. The “classname” attributes in keyLocator and tokenGenerator indicate the Java classes for the plug points. Most of the other attribute values are properties saying how to perform a specific task, i.e. signing, locating keys, or generating tokens. For example, since exc-c14n in canonicalizationMethod is located under signingInfo, it is used during the signature processing.

With the transformation sample shown in Figure 3, we can see how to derive configuration files like Listing 2 and Listing 3. We start from a security-annotated application model. Its actual representation is similar to the upper part of Figure 3. Meanwhile, the target configuration objects are equivalent to Listing 2 and Listing 3, in spite of their different representations, i.e. UML vs. XML. Thus, it is easy to generate most of the target configuration, i.e. DD and binding, by simply applying the transformation to the security-annotated application model.

However, some parameters in the target configuration are still empty with this transformation by itself. In Table 2, we pick up some parameters that are attribute values, as emphasized using bold in Listing 3, and they specify how substitution is performed. C14N and the digest methods are embedded in the transformation. On the other hand, Signing algorithm and Token type are specified with a platform model. In other words, ignoring such technical details, the transformations can have a broader scope of applicability. Actually, the transformations we discussed here can also be used for the signa-

Table 2. Substitution for Parameters in Binding

Parameter	Substitution
Signing algorithm	Platform Model
C14N method	Transformation
Digest Method	Transformation
KeyLocator	User defined
TokenGenerator	User defined
Token type	Platform Model
CallbackHandler	User defined

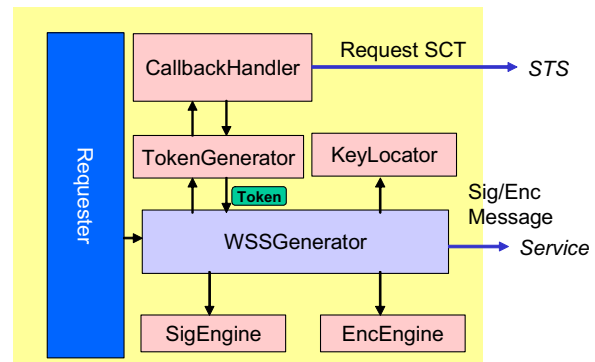


Figure 6. Architecture of WS-Security Handler in IBM WebSphere

ture with X.509 certificate. In that case, the platform model is much simpler than the model for WS-SK.

While we cannot automatically determine class names for the plug-in points (marked as “User defined” in the table), we can provide rich information on how the classes should be implemented. We can prepare a UML representation, capturing an architectural design of the WS-Security handler as in Figure 6. The details can be filled in from two information sources: the platform model and the generated configuration. Providing such detailed specifications for the plug-in point classes is one of our future projects.

5. Discussion

5.1 Related Works

Premkumar, et al. discusses security issues that we may want to address in the context of software engineering [5]. According to their term, what we address in this paper is considered as *post-deployment configuration management* (PDCM). While they describe why PDCM is a difficult task, we have provided a way to perform a PDCM task, using the WS-Security architecture. We also have shown that plug-in points must be considered when generating security configurations because of the extensibility of WS-Security. To the best of our knowledge, no work has been done for plug-in components. We believe there are some challenges in this area.

UMLSec [18] is an approach to include security in UML. Unlike our approach, the detailed security descriptions can be added to UML because they want to

generate secure code for applications. In our approach, we assume that applications that do not contain security code are deployed in some environments. Rather, security is configured during the deployments. We think that this situation will be more important in SOA environments.

SecureUML [19] is another approach to include security in UML. Addressing Role-Based Access Control (RBAC), users can add detailed information to UML. However, such detailed descriptions require users to deeply understand RBAC. While [19] provides an example to generate an EJB DD, we don't see the difference between SecureUML and the EJB DD in terms of abstraction levels. Rather, only the syntax is changed in the example.

In contrast with UMLSec and SecureUML, we use the idea of a *transformation* so that the security intent annotated by non-security experts is refined. As in our detailed example for WAS, the difficult part is to generate bindings. In order to enable the elaboration, we have introduced a *platform model* that is not considered in the existing models.

5.2 Remaining Issues

The target of our tool is people who are not security experts. We here discuss how we should enhance our tool from that perspective.

Models in MDA must be executable. When users add security intentions to UML, the model should be executable using our tool. In our current tool, the security intentions are added correctly, and thus the following generation processes can be performed. However, people who are not security experts want to know whether or not the annotated portions are correct. In order to enhance our tools, we need to identify a collection of security threats, and show that the threats are blocked with the annotated security intentions. In that sense, it is worthwhile for non-security experts to see the execution of a security-annotated UML model.

It is also important to take into account other security concerns such as access control and privacy. While we focus on WS-Security in this paper, it is possible to include other security features at a higher abstraction level. In order to define transformations for them, we have to consider how platform models should be represented. Since there are some exiting works for representing access control policy in UML as in SecureUML, we can leverage them. Once a platform model is defined, we can apply a similar method to generate detailed configurations for access control and privacy.

6. Concluding Remarks

It is becoming harder to configure security in the SOA environment because applications running on various security infrastructures must be coupled. Addressing such issue, we have proposed a MDA-based tooling

framework for WS-Security. As demonstrated in Section 4, it requires much information such security environment and WAS architecture, in order to refine abstract security intent into IT level artifacts. However, as MDA spreads out, most of such necessary information can be collected during the ordinary development process. In that sense, our tooling framework will be practical enough in a near future.

Since SOA assumes frequent changes in business and IT environments, security configuration should follow such changes. In order to address such issue, we believe that we should have enough information in an organized manner. With our tool, we can capture security artifacts at various levels. Although those artifacts are not sufficient, they are good start to consider. In addition, we think that such security artifacts can be used for checking security compliance.

References

- [1] A CBDI Report Series – Guiding the Transition to Web Services and SOA, http://www.cbdiforum.com/bronze/downloads/ws_roadmap_guide.pdf
- [2] Security in a Web Services World: A Proposed Architecture and Roadmap, Apr 7, 2002. <http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/>
- [3] International Telecommunications Union-Telecommunication Standardization Sector Recommendation X.509 "Information Technology--Open Systems Interconnection--The Directory: Authentication Framework". (Equivalent to ISO 9594-8.)
- [4] Jason Garman, Kerberos: The Definitive Guide, O'Reilly & Associates Inc., 2003
- [5] Premkumar T. Devanbu Stuart Stubblebine, Software Engineering for Security: a Roadmap, ICSE 2000.
- [6] Tatsubori, Imamura and Nakamura, Best Practice Patterns and Tool Support for Configuring Secure Web Services Messaging, ICWS 2004.
- [7] Web Services Security Policy Language (WS-SecurityPolicy), Dec 18, 2002. <http://www-106.ibm.com/developerworks/library/ws-secpol/>
- [8] Frankel, D: *Model Driven Architecture*, Addison-Wesley 2003.
- [9] Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) OASIS Standard 200401, March 2004. <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
- [10] IBM WebSphere Application Server <http://www-306.ibm.com/software/webservers/appserv/was/>
- [11] IBM Rational Application Developer <http://www-306.ibm.com/software/awdtools/developer/application/index.html>
- [12] Unified Modeling Language <http://www.omg.org/technology/documents/formal/uml.html>

- [13] Business Process Modeling Language (BPML), Business Process Management Initiative, Mar 8, 2001, <http://www.bpml.org/bpml.esp>
- [14] Simon Johnston, Modeling security concerns in service-oriented architectures, IBM developerWorks, 2004 <http://www-106.ibm.com/developerworks/rational/library/4860.html>
- [15] Takeshi Imamura and Michiaki Tatsubori: Patterns for Securing Web Services Messaging, *OOPSLA 2003 Workshop on Web Services and Service Oriented Architecture Best Practice and Patterns*, Anaheim, California, USA, November 26-31, 2003.
- [16] Web Services Security Kerberos Binding, IBM and Microsoft, 2003. <ftp://www6.software.ibm.com/software/developer/library/ws-seckerb/WS-Security-Kerberos.pdf>
- [17] E. Baize and D. Pinkas, "Generic Security Service Application Program Interface Version 2 Update 1," RFC 2743, January 2000.
- [18] Jan Ju"rgens, UMLsec: Extending UML for Secure Systems Development, Proceedings of the 5th International Conference on The Unified Modeling Language (UML2002), Dresden, Germany, September 30 - October 4, 2002. Lecture Notes in Computer Science Vol. 2460, pp. 412-425, 2002.
- [19] Torsten Lodderstedt, David A. Basin, and Ju"rgen Doser, SecureUML: A UML-Based Modeling Language for Model-Driven Security Proceedings of the 5th International Conference on The Unified Modeling Language (UML2002), Dresden, Germany, September 30 - October 4, 2002. Lecture Notes in Computer Science Vol. 2460, pp. 426-441, 2002.