

SOABSE: An Approach to Realizing Business-Oriented Security Requirements with Web Service Security Policies

Tan Phan, Jun Han, Ingo Mueller, Malinda Kapuruge

Faculty of ICT, Swinburne University of Technology,
Melbourne, Australia
{tphan, jhan, imueller, mkapuruge}@swin.edu.au

Steve Versteeg

CA Labs, Melbourne, Australia
{Steve.Versteeg}@ca.com

Abstract— A critical issue in developing Web Service-based business applications is the realization of business-level security requirements with system-level security mechanisms using the WS-* standards. Current practice has primarily relied on the engineer’s experience and lacks consistency and methodological support. This paper introduces an approach to Web Services security engineering called SOABSE, which systematically models, designs and implements security for a WS-based application from a given set of business-oriented security requirements. It includes 1) a stepwise process that systematically transforms business-level security requirements into system-level WS-* security policies, and relies on 2) a security realization model that maps business-level security objectives to WS-* security realization mechanisms and 3) a security deployment model that sets out the security-oriented Web Service deployment information. A prototype tool supporting the approach is also introduced.

Keywords: security attributes, WS-Security, security models

I. INTRODUCTION

Addressing security requirements in large SOA systems pose new challenges because of the distributed, inter-organizational nature of such systems [1]. Security requirements are not pure technical-level concerns as many security requirements come from business and legal requirements such as Sarbanes Oxley and Basel II Accord. An SOA system may operate in different contexts and countries and thus must adhere to different rules and regulations with regards to security of the system. Because of the context-sensitive nature of security settings for SOA systems, many aspects of security in Web Services (WS) (as the main implementation technology for SOA) are externalized from the application logic and are provided by WS container/infrastructure using declarative, policy-based “instructions”. Security settings for a WS-based application are represented in WS-SecurityPolicy and are applied (or attached) to various WS and related elements. Such policies are then enforced at runtime using WS-Security infrastructure provisions, which are configured with the application’s security context.

For WS-based SOA systems using WS-Security, ensuring traceability for security requirements in design and deployment is a major challenge. In particular, how to guarantee that WS security policies created by software

engineers at the system-level reflect the original business-level requirements remains an unsolved issue. This is because 1) there is the lack of a process that the engineers can follow to create WS security policies from business-level requirements and 2) WS security policies focus on “how” security measures are performed, not on “what guarantee” such measures provide. Moreover, current system development approaches typically integrate security into systems in ad hoc manners and only consider security at the deployment or system administration phase [2].

In this paper, we introduce a process-driven approach called SOABSE (SOA- Business Security Engineering) and related techniques to improve the current development practice of WS-based software systems. The process starts with the modeling of *security objectives* at business-level which are then systematically mapped into security measures performed on WS in the form of *WS security policies*. This systematic process and associated techniques provide better requirements traceability for security, assist business analysts in formulating security requirements and help software engineers in interpreting business requirements thus minimizing mistakes compared to handcrafting WS security policies.

Underpinning the SOABSE process are three general models that the analysts and engineers use. The *generic security model* sets out the security attributes and links them to the corresponding realization mechanisms (i.e., security functions and eventually security policies). For each WS-based application, an instantiation of the *security deployment meta-model* is made to systematically capture the application’s specific security context. An instantiation of the *business entity meta-model* for an application captures the business entities and their relationships, and provides the basis for stating business-level security requirements and relating WS security policies to WS elements.

This paper is organized as follows. Section 2 presents a motivating business scenario from which business-level security requirements are extracted. Section 3 presents the SOABSE framework including the process, the three general models, and the related transformation techniques illustrated by a running example. In section 4 a prototype tool supporting SOABSE is presented. Discussion of related work and evaluation of SOABSE are presented in sections 5 and 6 and the paper concludes in section 7.

II. A MOTIVATING EXAMPLE

In this section, we present a business scenario at a hypothetical multi-national bank named SwinBank, including a business process implemented in SOA. At SwinBank, when a Customer applies for a mortgage loan, a LoanOfficer accepts the application and triggers the bank's loan approval process. The bank arranges a professional appraiser to estimate the market value of the collateral property. Meanwhile, the customer's TaxFileNumber is forwarded to a CreditCheckingUnit to verify the CustomerCreditHistory. A list of credit scores from credit rating agencies is then obtained. The final step is to check the customer's RepaymentCapacity by assessing the income against the amount to be repaid and a decision is made whether to approve the loan.

A WS-based implementation of the business process called SwinMortgage is being conducted at SwinBank. The system is made up of a Web front-end called SwinMortgageWeb for the LoanOfficer to use, a BPEL business process called SwinMortgageBPEL implementing the loan processing procedure, and a number of WSs for the processing activities. Those include a third party PropertyAppraiser and CreditRater services and an internal RepaymentCapacityAnalyser service.

Applicable Rules, Regulations and SwinBank Business Policies. The operation of SwinBank is subject to many rules and regulations, in particular, the Australian Privacy Act 1988 (APA) [3], Sarbanes-Oxley 404, and Basel II. Here we take the APA as an example. The following are some security-oriented requirements identified from APA.

1. APA principle 6: A record-keeper who has possession or control of a record that contains personal information shall take such steps ... to ensure:
 - *that the record is accurate (R1).*
2. APA principle 4: A record-keeper who has possession or control of a record that contains personal information shall ensure:
 - *that the record is protected ...against unauthorized access, use, modification or disclosure, and against other misuse (R2); and*
 - *that ... everything reasonably ... is done to prevent unauthorized use or disclosure of information contained in the record (R3).*

Following the ISO/IEC TR 17944 [4] guideline for security practices in banking and financial systems, SwinBank also mandates the business requirement below.

- *Loan applicants must not be able to repudiate the lodgment of a loan and the bank must not be able to repudiate the receipt of a loan application (R4).*

The above security-oriented business and legal requirements (**R1-R4**) must be addressed in SwinMortgage

and because of their potential changes and context-sensitivity it is desirable that the requirements are externalized from the functional logic of SwinMortgage. For that reason, the development team of SwinMortgage decides that the security aspects related to service interactions and message protection will be provided through a WS Security infrastructure such as Apache Rampart which is engaged with the Application Server where the WS and applications are deployed.

The task now is to define WS security policies to address the original business requirements. A challenge in doing so is to ensure that all relevant business requirements are properly stated, interpreted and implemented by SwinMortgage developers. Furthermore, manually creating WS security policies and using WS-PolicyAttachment is tedious and error-prone. To address these issues, a process and associated techniques are required that systematically support the modeling, design and implementation of security for WS-based software systems from a set of business requirements.

III. SECURITY MODELING, DESIGN AND DEPLOYMENT

We present in this section the SOABSE security engineering process for WS-based systems. We start by presenting an overview of the process followed by a detailed discussion of each step and the models and techniques used.

Fig. 1 shows an overview of SOABSE. It is based on the conventional software development process (for functionality), but incorporates in parallel the additional aspects relating to security-oriented modeling, design and deployment. Security is not considered only a system-level concern and should not be considered an "afterthought" when the functional development is completed. Instead, it should start with the identification and modeling of security requirements at the business analysis phase and continue throughout the system development lifecycle.

The business analysis phase for an application results in (among other artifacts) a *business entity model* capturing the *business entities* and *their relationships* relevant to the application. The security-oriented business policies are formulated as a set of *business security objectives* relative to the business entities. At the design phase, *abstract service elements* such as abstract WSs and BPEL processes are identified and are annotated with entities from the *business entity model*. Based on the annotation, *service security objectives*, which apply *security attributes* on the identified *abstract service elements*, are derived from the *business security objectives*.

At the implementation phase, a security deployment model for the application is developed following the security deployment meta-model to set out the security configurations for the infrastructure support. The *service security objectives* are translated into *security functions* to be performed on the service elements, using the *generic security model* with reference to the *security deployment model*. These security functions are then mapped to WS-SecurityPolicy configurations for use at runtime by the WS-Security infrastructure to provide security enforcement

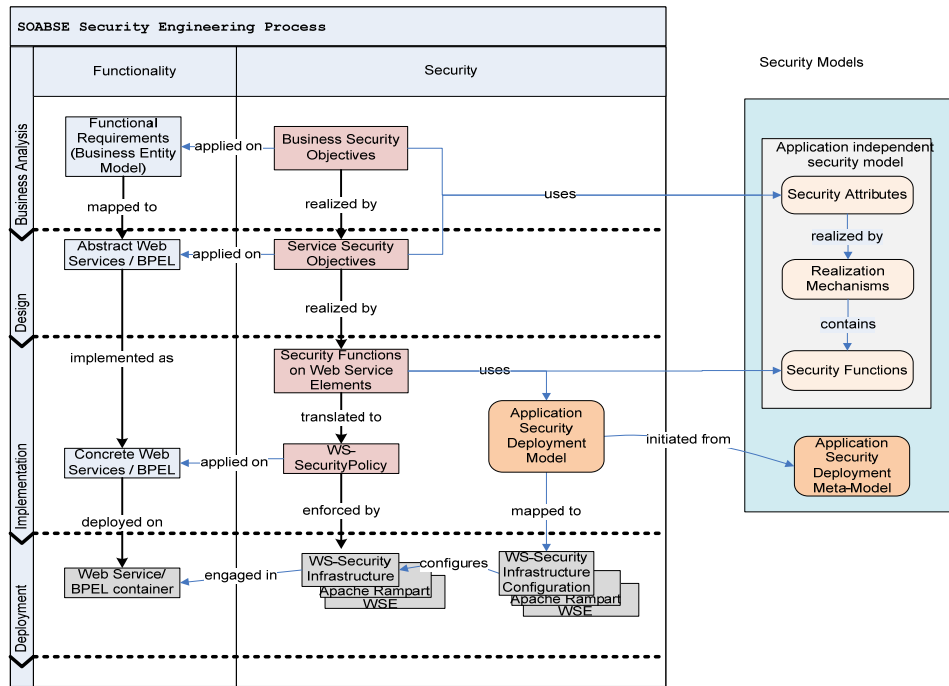


Figure 1. SOABSE Web Services security design process

on top of the application's WS implementation. Details of the *security deployment model* are also translated into configuration for the security infrastructure. Details of each step and the models and techniques involved are discussed in the following subsections.

A. Identifying Business-Level Security Objectives

At the business analysis phase we advocate the modeling of business entities in the application through an application *business entity model*. This model is created based on an analysis of the functional requirements to extract the relevant "entities" (which are typically nouns in the business requirements). We expect that this model will serve as the starting point for the creation of ER or UML diagrams in the design phase.

A *business entity* in the application's *business entity model* represents a business-oriented concept from the application. It can either be 1) a *processor*: performing business logic at request (e.g., LoanProcessor); 2) a *dataItem*: holding business data (e.g., CustomerTaxFileNumber); or 3) a *userRole*: representing a user role (e.g., LoanOfficier) in an organization, having access to *dataItems*, and being able to ask *processors* to perform actions. In an SOABSE *business entity model*, each entity is a direct or indirect specialization of one of those three basic entity types. Using this approach, applications can be viewed as compositions of interacting entities. We argue that such a model is simple and intuitive enough for use by business analysts.

The entities in the SwinMortgage application include 1) *dataItems*: LoanData, CreditHistory and TaxFileNumber; 2) *processors*: PropertyAppraiser,

CreditChecker, RepaymentCapacityAnalyser, and SwinMortgageBPEL; 3) *userRoles*: LoanOfficier. Part of the business entity model for SwinMortgage can be seen in Fig. 2.

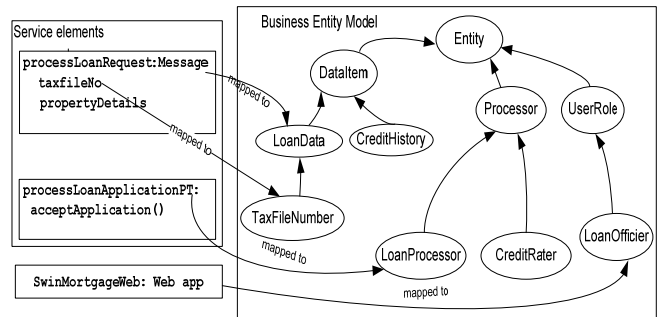


Figure 2. Part of SwinMortgage's business entity model and mapping to Web Services elements

Based on an application's *business entity model*, *business security objectives* can be formulated to model and identify the application's business security requirements. A *business security objective* is specified in the form of *securityAttribute[businessEntity]* meaning the security guarantee implied by the *securityAttribute* (such as *confidentiality*) must hold on the *businessEntity* (e.g., CreditHistory). There are seven common *security attributes*, which are derived from the Common Criteria for IT Security Evaluation [5]. Such security attributes are essential for securing WS messages: *confidentiality*, *integrity*, and *freshness*; and interactions: *audit*, *authentication*, *authorization* and *non-repudiation*. To

identify business security objectives, the analyst can use security requirement elicitation techniques [6] or the Unified Compliance Framework (www.unifiedcompliance.org).

Note that in principle, such security protection can be provided by a WS infrastructure given the necessary specifications in WS-SecurityPolicy. In this approach, we do not consider security attributes that are related to the persistence of data in a database or the processing of data inside service implementation as such features often need to be mixed with functional logic of the application and are not supported by WS-Security.

Table 1 presents a subset of *business security objectives* for SwinMortgage corresponding to the business requirements (R1-R4) identified earlier.

TABLE I. BUSINESS REQUIREMENTS AND BUSINESS SECURITY OBJECTIVES FOR SWINMORTGAGE

Business Requirements	Business Security Objectives
(R1)	<i>integrity</i> [TaxFileNumber], <i>integrity</i> [CreditHistory]
(R2)	<i>authorization</i> [LoanOfficier]
(R3)	<i>confidentiality</i> [TaxFileNumber], <i>confidentiality</i> [CreditHistory]
(R4)	<i>non-repudiation</i> [LoanData]

B. Deriving Web Service Security Objectives

Service Design and Annotation. To progress from the business analysis phase to the design phase and translate business security objectives into WS security objectives, we first annotate the *abstract service elements* for the application (as they are designed by the engineers) with the corresponding *business entities* from the application's *business entity model*. DataItems can be mapped to WS messages or their parts; processors can be mapped to WSs, their portTypes or operations (or BPEL processes that are exposed as WSs); userRoles are mapped to applications that the users interact with. The mapping is many-to-many as a business entity can be refined into multiple abstract service elements and a service element might at the same time represent multiple business entities. Part of the mapping for the SwinMortgage example can be seen in Fig. 2 in the previous sub-section.

The mapping annotation can be in the form of a custom mapping table or other formats. We recommend WSDL-S [7] for its simplicity, popularity and standard-compliance. The annotation serves two purposes, it first improves the traceability of service design decisions and secondly it enables the automatic derivation of *service security objectives* from the *business security objectives*

C. Configuring Security Deployment

Following the design phase, new services are developed or existing services are leveraged. Once the services and related elements are created, developers need to come up with a deployment model to prepare them for deployment. At present, such deployment model (e.g. UML deployment diagram) typically concerns the functional aspects of the system including topology of services, service composition

and clients, the endpoints where services are deployed and communication protocols (SOAP/HTTP for instance) they use. We argue that such model should also be extended to systematically capture deployment configurations concerning security and other non-functional properties. For the moment, there is the lack of a meta-model for security-related deployment information for an application. Therefore, different implementations of WS-Security such as Apache Rampart, Oracle Application Server and WSE use their own methods of configuring application-specific security infrastructure information, typically by utilizing custom *config* file with different details and schemas. This hinders, among other things, the substitutability between different WS-Security implementations.

We thus propose a *meta-model* for *security deployment* that includes a set of notations and concepts to provide a systematic and standardized way for representing the security characteristics of a SOA system in deployment. Fig. 3 shows the meta-model while Fig. 4 presents an instantiation of the meta-model for SwinMortgage. For each application, an instance of the meta-model is created to model how each service element is individually secured and how they securely communicate with each other. In particular, the model focuses on 1) the type of trust tokens associated with each service elements and 2) the trust relationships between service elements and the message flow between them.

As can be seen in Fig. 3, the meta-model comprises of two main types of concepts: 1) system elements and their relationships and 2) a hierarchy of security tokens owned by such elements to protect the system (shaded ellipses). The key elements in the model are as follows.

Security entities (se): are services and other software components in an SOA system. Security entities can own security tokens, have trust relationships with other entities and can interact with others by sending and receiving messages. For example, SwinMortgage (see Fig. 3) has five security entities (denoted $\langle\langle se \rangle\rangle$) including one Web application (SwinMortgageWeb), one BPEL business process (SwinMortgageBPEL) and three WSs.

Messages (m): define the set of messages such as *processLoanRequest* that travel between security entities. Messages can comprise of different parts. There are four pairs of messages in SwinMortgage including, for example, *checkCreditHistoryRequest/Response* (denoted M_{Req3} , M_{Res3} in Fig. 4).

Security channels (sc): between security entities denoted $SC_{X \rightarrow Y}$ is a uni-directional binary relationship (e.g., $SC_{X \rightarrow Y}$ denotes a channel from X to Y). In SwinMortgage we have eight security channels, including $SC_{SwinMortgageWeb \rightarrow SwinMortgageBPEL}$ (see Fig. 4).

Message flow (mf): $SC_{X \rightarrow Y} \{m_1, \dots, m_n\}$: defines the set of messages $\{m_1, \dots, m_n\}$ that travel on the channel $SC_{X \rightarrow Y}$. An example flow in SwinMortgage (see Fig. 4) is $SC_{SwinMortgageWeb \rightarrow SwinMortgageBPEL} \{processLoanRequest\}$.

Security tokens (*st*): define the types of security tokens available in a specific security context. These tokens can include (1) *individual tokens* (i.e., *public and private key pairs*), such as $X509\{K_0, -K_0\}$ in Fig. 4; (2) *symmetric tokens* (i.e., a shared key between two entities), such as $K_{\text{SwinMortgageBPEL, PropertyAnalyserService}}$ (denoted as $\langle\langle\text{shareKey}\rangle\rangle$ in Fig. 4); or (3) *asymmetric tokens* (i.e. a token, typically a pair of *username/password*, of one entity on another entity), such as $K_{\text{SwinMortgageBPEL-}}\text{RepaymentCapacityAnalyzingService}$ (denoted as $\langle\langle\text{hasUserNameOn}\rangle\rangle$ in Fig. 4).

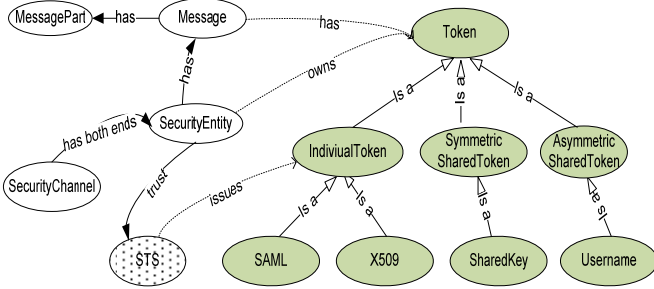


Figure 3. Security deployment meta-model

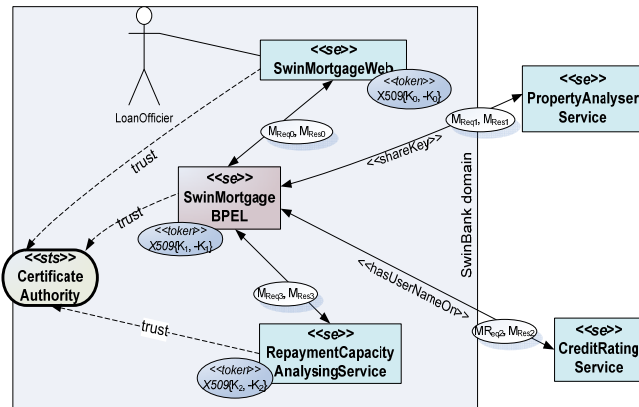


Figure 4. SwinMortgage's security deployment model

D. Realizing Security Objectives with Security Functions

This section presents the procedure and techniques involved in transforming *service security objectives* into a set of *security functions*. This essentially transforms high-level security requirements (*what* need to be guaranteed) into system-level mechanisms which realize such requirements (*how* the requirements are achieved). The transformation relies on a set of security patterns and best practices that we organize into the *generic security model*. Below, we first discuss the model and then the transformation procedure.

Generic Security Model: Security Attributes and Realization Mechanisms. The *generic security model* can be used across different applications. It maps each *security attribute* (discussed in section 3A) to one or more *security realization mechanisms*, each of which is a sequence of *security functions* that can be performed by some *function*

performer to realize the security objective. A *function performer* can be a message *sender* or *receiver*, and represents the party that performs the function in a message exchange. Realization mechanisms for each attribute are drawn from various security patterns [8, 9], standards [5] and best practices [10].

A *security function* is characterized by a function name and a set of *function properties* which are pairs of *property name* and *property value*. From analyzing [8, 9],[10] and WS-Security, we have identified the following security functions: *encrypt()*, *decrypt()*, *sign()*, *verifySignature()*, *log()*, *attachTimestamp()*, *verifyTimestamp()*, *attachUsernameToken()*, *verifyUsernameToken()*, *attachUserRightToken()*, *verifyUserRightToken()*, *attachBinaryToken()*, *verifyBinaryToken()*. We discuss here as an example the function *sign()*.

The *sign()* function involves generating a hash of the message and encrypting it to generate a digital signature, using a key associated with the message sender. It has four properties which are its *binding* method which can be symmetric (using a shared secret) or asymmetric (using sender's key pair); the *algorithm* for generating the digest (typically SHA); the *algorithm* for creating the signature (typically HMAC, RSA or DSA); and the *token* used for generating the digest and signature (this is application-specific and is provided by the *security deployment model*). This can be represented as

```

sign(binding={symmetric|asymmetric},
digestGenerationAlgorithm={SHA},
sigGenerationAlgorithm={HMAC|RSA|DSA},
signingToken=app-specific)

```

An overview of the WS security model is presented in Fig. 5. In this figure, we omit the details of *function performer* and some of the verification functions (such as *verifyTimeStamps()*) for brevity. Note that even though the *generic security model* is *generic*, developers can customize the model to suit their organization or project's requirements. Such customization is in the form of selecting the preferred *realization mechanism* for each security attribute and selecting the preferred *property value* for each property. For example, SwinMortgage follows the ISO/IEC TR 17944 [4] standard and WS-I Basic Security Profile [11] and uses *asymmetric* binding for signing and RSA as the *sigGenerationAlgorithm*. Details of each *security attribute* and its *realization mechanisms* are explained below.

The format *a:rm*, where *a* is the security attribute and *rm* is the realization mechanism that can be used to achieve *a*, is used. When there exist multiple *rm* for one *a*, we use the format *a:XOR(rm₁,...,rm_n)* to denote one exclusive choice of a *rm* (out of *n* possible *rm*) for each *a*. Each *rm* is represented in the format $\langle p_1.f_1, \dots, p_n.f_n \rangle$ denoting a sequence of *security function* *f_i* performed by performer *p_i* {*i*=1...*n*} respectively for that realization mechanism.

Message Confidentiality: ensures messages can be read only by intended recipients, and is achieved using message encryption at sender and decryption at receiver.

confidentiality: $\langle \text{sender.encrypt}(), \text{receiver.decrypt}() \rangle$

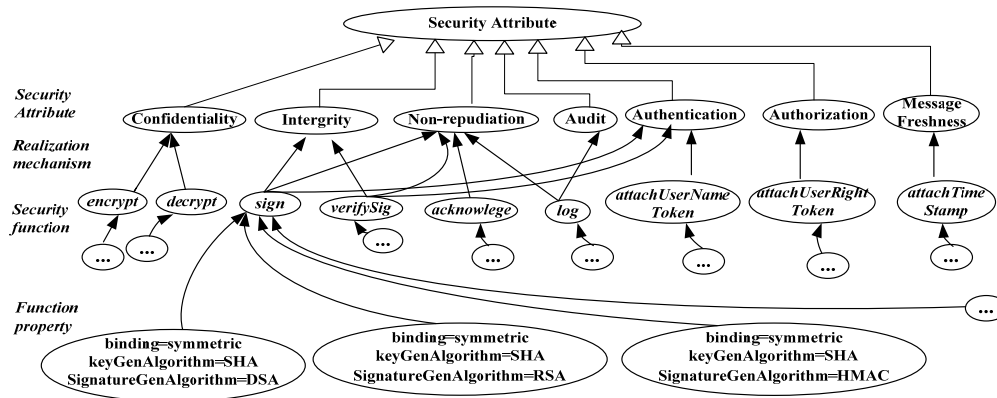


Figure 5. A segment of Web Services security model

Message Integrity: ensures messages are not tampered with while in transit, and is achieved using digital signature (signing) at the message sender end and signature verification at the receiver end.

integrity: <sender.sign(), receiver.verifySig(>

Non-repudiation: ensures the sender of a message cannot repudiate that he has sent the message, and the receiver cannot repudiate that he has received the message. To achieve this, the sender must first sign() the original message using his private key (*asymmetric binding*). Upon receiving the message the receiver must log() the received message and acknowledge() the receipt of the message by sending back a signed receipt to the sender. The sender needs to log() that acknowledgement for proof that the receiver has actually received the message. For the moment, WS-Security does not fully support non-repudiation.

non-repudiation: <sender.sign(binding=asymmetric), receiver.log(), receiver.acknowledge(), sender.log(>

Authentication: ensures message sender is the one he claims to be and is achieved by attaching an appropriate security token, trusted and verifiable by the receiver. Such tokens can be an encoded X.509 Certificate (*binaryToken*), or a username token with password (*usernameToken*). This can also be achieved using mechanisms such as digital signature (*sign()* and *verifySig()*).

authentication: XOR(<sender.sign(), receiver.verifySig(>, <sender.attachUserRightToken(), receiver.verifyUserRightToken(>, <sender.attachBinaryToken(), receiver.verifyBinaryToken(>)

Authorization: ensures services can only be accessed by principals with appropriate rights. This can be achieved by the sender attaching a security token, trusted by the receiver, to the service invocation message indicating his roles and credential. The receiver then needs to verify such token. The pair of *attachUserRightToken()* and *verifyUserRightToken()* are used for this purpose.

authorization: <sender.attachUserRightToken(), receiver.verifyUserRightToken(>

Audit: ensures that messages sent and received are recorded for later analysis. This is achieved by logging both

the sending and the receiving of the message at the sender and receiver ends respectively.

audit: <sender.log(), receiver.log(>

Message freshness: ensures that the message received by the receiver is *current* (i.e. not an old message captured and later resent by an intermediary). This is achieved by attaching a *timestamp* into the message and then *signing* it. The signature and the timestamp are then verified by the receiver.

message freshness: <sender.attachTimeStamp(), sender.sign(), receiver.verifySignature(), receiver.verifyTimeStamp(>

Transformation Procedure. Fig. 6 presents the procedure for transforming a set of *service security objective* into a set of security functions performed by the relevant entities to realize the objectives. A visualization of the transformation for the service security objective *non-repudiation*[processLoanRequest] is given in Fig. 7.

The transformation procedure takes a set of *service security objective* *S* and returns *A* - a mapping of a set of WS elements *w* to a set of security functions *G*, with property values fully specified, that need to be performed by each *w*. It has two major steps 1) identify a mapping of WS elements *w* and the set of security functions that need to be performed on them (Line 2-16, and 2) for each security function to be performed on an element, identify the values for their function properties. The first step is automatic and the second step can be automatic or interactive as required.

In the first step, an initial mapping with keys being abstract WS elements (set of *w*) and values being empty (line 2) is declared. The procedure then loops through all *service security objectives* (line 3). For each *security objective* *s_i* with security attribute *a_i* (such as *non-repudiation*) and WS element *w* (such as *processLoanRequest*) it looks up the *generic security model* to identify the corresponding *realization mechanism* *m_j* for *a_i* (in this case a sequence of <sender.sign(binding=asymmetric), receiver.log(), receiver.acknowledge(), sender.log(> (see above) (line 4).

```

Inputs:
 $S = \{s_1, s_2 \dots s_n\}$ , the set of security objectives
  where  $s_i = (w_i, a_i)$ ,  $w_i$  is the WS element, and  $a_i$  is the security attribute
 $M$ , the set of mappings  $\{(a_1 \mapsto m_1), (a_2 \mapsto m_2) \dots (a_k \mapsto m_{|M|})\}$  from security attribute  $a_j$  to
security realisation mechanism  $m_j$ 
  where  $m_j$  is the set of performing tuples,  $\{(f_{jk}, p_{jk}), \dots\}$ 
  where  $f_{jk}$  is the function, and  $p_{jk}$  is the performer
 $SDM$ , the application's security deployment model
Output:
 $A$ , the set of mappings  $\{(w_1 \mapsto G_1) \dots\}$  from each WS element  $w_q$  to a set of applied functions
 $G_q = \{f_{q1}(w_{q1}), f_{q2}(w_{q2}) \dots\}$ 
1 begin
2   initialise  $A$ , so that each WS element maps to  $\emptyset$ 
  // iterate over security objectives
3   foreach security objective  $(s_i = (w_i, a_i)) \in S$  do
4     get srm  $m_j$ , such that  $(a_i \mapsto m_j) \in M$ 
5     foreach  $(f_{jk}, p_{jk}) \in m_j$  do
6       // identify WS elements  $E$ , to perform function
7       switch type of  $w_i$  do
8         case WSDLMessage or WSDLMessagePart
9           Identify security channel,  $c$ , which  $w_i$  travels on
10          if  $p_{jk}$  is a sender then set  $E$  to contain sender on  $c$ 
11          else set  $E$  to contain receiver on  $c$ 
12          case WSDLPortType, WSDLOperation or WSDLService
13            set  $E = \{w_i\}$ 
14            otherwise
15              //  $w_i$  is a client application
16              set  $E$  to be WSDLServices that  $w_i$  interacts with
17          get  $G_q$ , such that  $(w_q \mapsto G_q) \in A$ 
18          foreach  $w_q \in E$  do set  $G_q = G_q \cup \{f_{jk}(w_q)\}$ 
19          // Set values for function properties
20          foreach  $(w_q \mapsto G_q) \in A$  do
21            foreach  $f_{qp} \in G_q$  do
22              // Each property in property set of the function
23              foreach  $prop_{qpr} \in f_{qp}$  do
24                if value for  $prop_{qpr} \in SDM$  then use such value
25                else set  $prop_{qpr}$  to highest preference of all available values
26          end
27        end
28      end
29    end
30  end
31 end

```

Figure 6. Transformation procedure

For each pair of $\{\text{performer } p_{jk} \text{ and function } f_{jk}\}$ in m_j we then map the *performer* to a concrete WS or client application (security entities) that performs it. The mapping is determined as follows. If *wse* is a message or a message part (line 7), the application's *security deployment model* is looked up to identify the *security channel* on which *wse* is delivered (line 8). If *performer* is the message sender it is set to the service/application which is the sender end of the channel (line 9). Similarly, if *performer* is the receiver then it is set to the receiver service/application of the channel (line 10). For example, for the pair *sender.sign()* to be performed on *processLoanApplication* in the example above, by looking up *SwinMortgage security deployment model* (Fig. 4) it is known that *processLoanRequest* travels on the channel between *SwinMortgageWeb* and *SwinMortgageBPEL* thus *sender* corresponds to *SwinMortgageWeb*. We then add *sign[processLoanRequest]* to the set of functions performed by *SwinMortgageWeb* (line 17). If *security*

objectives are specified on services, portTypes, operations or service clients, the security functions to be performed on them and the function performer are identified in a similar manner (line 11-14).

In the second step, we assign properties values to each of the identified function (line 17-21). This step loops through all the functions f_{qp} to be performed on each service element w_q and, for each function, loops through all the properties $prop_{qpr}$ of the function. If the property's value is set by the application's *security deployment model* (*SDM*), it is looked up from such model, passing information about the performer of the function (line 20). For brevity, we do not discuss the logic of this look up. For example, it is known from the previous step that the function *sign()* is performed by *SwinMortgageWeb* which owns an X.509 certificate thus the value of the signing token can be set to the private key of *SwinMortgageWeb* endorsed by the certificate. If the property's values are specified in the *generic security model* instead, the preferred value for the property is selected (line 21). For example, in *SwinMortgage*, out of three possible

values for *digestGenerationAlgorithm*, SHA is preferred (as discussed previously) and it thus becomes the selected value.

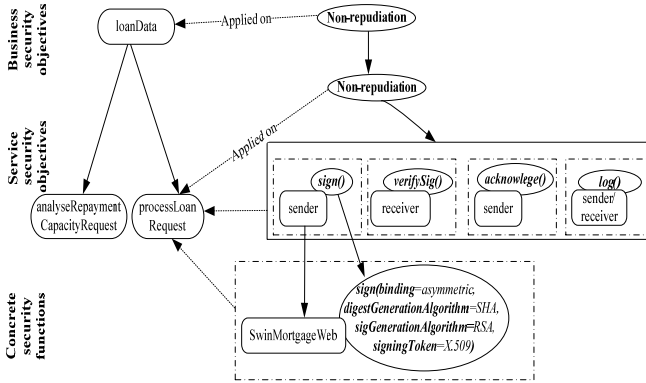


Figure 7. Example transformation

Using this procedure, an example *sign()* function's property values can be initialized as
sign(binding= asymmetric,
digestGenerationAlgorithm=SHA,
sigGenerationAlgorithm=RSA, signingToken=X.509)
The function is then performed by SwinMortgageWeb on processLoanRequest (see Fig. 7).

The procedure presented above assumes that the *generic security model* is customized (i.e. when there exists multiple *realization mechanisms* for a *security attribute*, a preferred one is specified and when there exist multiple potential values for a *security property*, a preferred value is specified). In such a simple case, this step can be automated. In a more complex case where the preferences are not pre-specified, this is an interactive step which involves the developer selecting their preferred realization mechanisms or property values.

E. Generating WS-SecurityPolicy

Once the *security functions* are generated from the security objectives, they are mapped into WS security policies. Due to space limitations, we do not discuss the full mapping, a summary is presented below.

1. Functions are mapped to WS-SecurityPolicy protection or binding assertions

TABLE II. FUNCTION MAPPING

Function	WS-SecurityPolicy Protection and Binding Assertion
<i>Encrypt()</i>	<sp:EncryptedElement>
<i>Sign()</i>	<sp:SignedElement>
<i>Acknowledge()</i>	<sp:SignedElement>
<i>Log()</i>	<sp:LogElement> This assertion has not been defined
<i>AttachUsernameToken()</i>	<sp:UsernameToken>
<i>AttachUserRightToken()</i>	<sp:SAMLToken>
<i>AttachTimeStamp</i>	<sp:Timestamp>

2. Function properties are mapped to binding assertions

TABLE III. FUNCTION PROPERTY MAPPING

Properties	WS-SecurityPolicy Binding Assertion
Binding value = symmetric	<sp:SymmetricBinding>
Binding value = asymmetric	<sp:AsymmetricBinding>
Binding value = hybrid	<sp:HybridBinding> (not defined in WS-Security.)
Algorithm	<sp:AlgorithmSuite>
Key	<sp:KeyInfo>

The excerpt below shows the skeleton of the generated WS-SecurityPolicy fragment for the function *sign()* applying on *processLoanRequest* to be performed by SwinMortgageWeb as discussed in section 3D above.

```

</SignedElements>
  <XPath>processLoanRequest</XPath>
</SignedElements>
<sp:AsymmetricBinding>
  ...<sp:ProtectionToken> ...
    <sp:RecipientToken>
      ...<sp:X509Token.../>...
    </sp:RecipientToken>
    <sp:InitiatorToken>
      ...<sp:X509Token.../>...
    </sp:InitiatorToken>
  ...</sp:ProtectionToken>
  ...<sp:AlgorithmSuite>
    ...<sp:Rsashal/>...
  </sp:AlgorithmSuite>
...</sp:AsymmetricBinding>

```

As can be seen in the excerpt, *sign()* is mapped to the WS-SecurityPolicy protection assertion *signedElements* configured to be performed on the *processLoanRequest* message. We assume *asymmetric* binding, *keyGenAlgorithm=SHA*, *signatureGenAlgorithm = RSA* (mapped to *Rsashal* algorithmSuite value). X.509 certificates are used as security tokens as discussed in the previous section, which are translated into *protectionAssertion* assertion details. This policy is then associated with the WS-Security infrastructure (e.g. Apache Rampart) where SwinMortgageWeb is deployed.

IV. PROTOTYPE TOOL

We have implemented a prototype development tool supporting SOABSE in JEE. It incorporates the *generic/meta-models* for use by business analysts and engineers. The prototype is designed as a multi-tier web application backed by a database. It comprises of two major tools, the *business analyst tool* and the *software engineer tool* (Fig. 8). It also implements the TransformationEngine and WS-SecurityPolicyGenerator. The meta-models and application-specific models and artifacts are stored in the database.

The business analyst tool allows a business analyst to define an application's *business entity model* and specify security objectives on its business entities. Via a web

interface (see the left side of Fig. 9) a business analyst can formulate *business security objectives* by ticking a set of checkboxes corresponding to the desired security attributes and business entities. The software engineer tool allows a software engineer to annotate system-level services with business concepts (see the right side of Fig. 9), customize the *generic security model*, and instantiate the *security deployment model*.

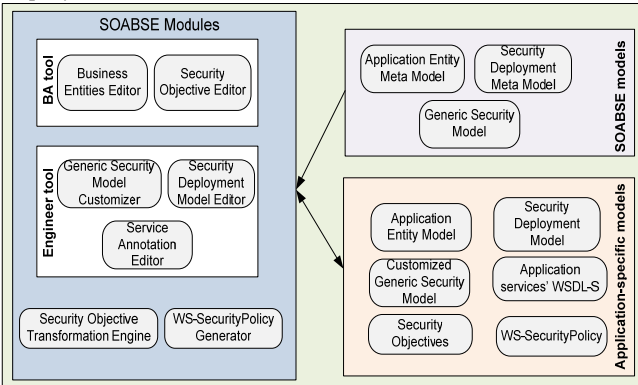


Figure 8. SOABSE prototype's conceptual architecture

Entity / Ouality	confidentiality	integrity	non-repudiation	freshness
TaxFileNumber	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CreditHistory	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
LoanData	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
LoanOfficer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
LoanProcessor	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
PropertyAppraiser	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CreditRater	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 9. Editing quality objectives (left) and annotating services (right)

The SOABSE TransformationEngine takes a set of security objectives and queries all the models for information and then uses the transformation procedure to convert the quality objectives into a set of security functions. The WS-SecurityPolicyGenerator then maps such functions into WS security policies.

In the prototype, all models are stored in the MySQL SOABSE_DB database. Database tables are mapped to object oriented models in Java using the JPA Object-relational mapping technology. The application is hosted using a Glassfish application server.

V. RELATED WORK

There have been a number of efforts in leveraging model-driven techniques for security engineering, starting with SecureUML [2] and UMLSec [12]. The application of such techniques into WS security modelling has been explored in work such as [13]. These approaches allow the creation of security-aware applications in an early phase of

system development. Unlike our approach, the models in [2, 12,13] stay at the system-level, which hinders the involvement of business analysts in the modelling. Moreover, while [2] focuses on role-based access control, [13] focuses more on confidentiality and integrity and neither provides a comprehensive model for other security qualities. In [14], the authors proposed a model-driven approach for annotating security requirements in business processes and transforming them into WS-SecurityPolicy (for authentication, integrity, and confidentiality) or XACML (for authorization). In such work, security experts create platform-independent security models for security requirements in an application, which are then mapped to various platform configurations. However, similar to [13], users [14] are security experts who model security requirements at the architectural level and thus there is still the requirement traceability gap between the business and architectural level.

In the area of SOA security engineering, the Sectet framework [15] provides a language to model security requirements for inter-organizational workflow. The model concerns three basic security qualities (confidentiality, integrity and non-repudiation) and some “advanced” security issues (access control, usage control, and custom “domain principles”). Such quality objectives are assigned to nodes in a distributed system and are enforced using the Sectet reference architecture. Unlike our transformation procedure, Sectet’s transformation process is mentioned very briefly and it is unclear how the abstraction is concretized (for example, how confidentiality is realized at the system-level).

There is also a body of work that performs formal analysis on WS security. In [16], a WS security verification tool has been created that allows the translation of security configurations in WS-SecurityPolicy into the TulaFale language which are then verified for correctness using formal logics-based reasoning. Such techniques can be used to verify the WS-SecurityPolicy generated by our approach.

In our previous work [17], we have proposed a general HOPE framework addressing the issue of aligning business requirements with system-level implementation of non-functional properties. In this work, we focus on addressing the requirement traceability of security-oriented development. The proposed SOABSE process and related modelling and transformation techniques substantially extend HOPE to enable the systematic analysis, design and deployment of security for WS-based SOA systems.

VI. DISCUSSION

In this section we discuss the advantages and limitations of the SOABSE approach. Our approach can improve the current WS security development practice with requirement traceability. This can be achieved because in SOABSE a systematic process is applied and security requirements are modeled at the very first phase of system development,

which are then semi-automatically transformed to system-level realization mechanisms.

Other contributions of the approach are the *generic security model* that can be applied to different WS-based applications with minimal customization, and the *security deployment meta-model* that provides a systematic way to represent an application's security deployment characteristics. The *generic security model* is able to capture all the relevant common security objectives as defined in the Common Criteria [5] and their realizations in the form of security functions. We have also provided a mapping of the generic model to WS-SecurityPolicy and a mapping of the *security deployment meta-model* to the Apache Rampart configuration schema, which is not further discussed due to lack of space.

One of the main limitations of SOABSE is the set of security aspects supported are limited to what can naturally be supported by a security infrastructure, in this case WS-* stack and thus SOABSE cannot support security features which need to be realized programmatically in WSs. Moreover, as shown in Table 3 and Table 4, WS-SecurityPolicy currently does not support all identified security attributes and their realization mechanisms. Another major limitation of the WS-SecurityPolicy model is that it presents a local viewpoint of security (what needs to be performed by individual security entities in a system) and lacks a system-wide global security view (how security actions and steps are coordinated among different entities to achieve a common goal). To overcome this, a system-wide security coordination model/language is needed.

The SOABSE development process also poses some additional overheads compared with current development methods, particularly the need to model a *business entities model*, *business security objectives*; the need to customize the *generic security model* and define the *application-specific security model*. However, we argue that the information captured in such models is necessary for the development of SOA security following any development process. At present, while not being modeled explicitly, they are still implicitly captured and represented. Besides, in this paper, we assume we would be able to specify WS security policies for all the services involved in the system. In reality, for services that exist outside the organization, we do not have control over their security settings and thus, what we need to do instead is to create "required" services from the security perspective and use a policy-based discovery mechanism to locate actual services satisfying the requirements [18].

VII. CONCLUSIONS

In this paper, we have presented the SOABSE framework that, in a systematic manner, assists practitioners in defining business-oriented security requirements and transforming them into system-level WS security policies for realization in service-based applications. SOABSE includes a step-wise security development process which can be used in parallel

with the traditional functional logic development process. A generic security model linking security attributes with their realization mechanisms and a meta-model for capturing application-specific security characteristics are introduced in the framework for use by the process. SOABSE improves security requirements traceability and minimizes human errors in incorporating security provisions into SOA systems. In future work, we will conduct an extensive case study to validate the applicability and scalability of the proposed approach and will investigate a mechanism to coordinate security settings from different endpoints to achieve system-wide security.

REFERENCES

- [1] M. Papazoglou, *Web services: Principles and technology*: Addison-Wesley, 2008.
- [2] T. Lodderstedt, D. Basin, and J. Doser, "SecureUML: A UML-Based Modeling Language for Model-Driven Security" in *«UML» 2002 — The Unified Modeling Language*, pp. 426-441, Berlin/Heidelberg: Springer, 2002.
- [3] Commonwealth of Australia, "Privacy Act 1988," available at http://www.austlii.edu.au/au/legis/cth/consol_act/pa1988108
- [4] ISO/IEC-15408, "Banking — Security and other financial services — Framework for security in financial systems" ISO, 2002.
- [5] ISO/IEC-15408, "Common Criteria for Information Technology Security Evaluation," <http://csrc.nist.gov/cc/>, ISO, 1999
- [6] C. Gutierrez, E. Fernandez-Medina, and M. Piattini, "Web Services-based Security Requirement Elicitation," *IEICE - Transactions on Information and Systems*, vol. E90-D, no. 9, pp. 1374-1387 2007.
- [7] R. Akkiraju, J. Farrell, J. Miller *et al.* "Web Service Semantics - WSDL-S," W3C standard submission, 2007; <http://www.w3.org/Submission/WSDL-S/>.
- [8] J. D. Meier, A. Mackman, M. Dunner *et al.*, "Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication," *Microsoft Patterns and Practices, MSDN library*, Microsoft Corporation, revised version 2006.
- [9] J. D. Meier, A. Mackman, M. Dunner *et al.*, *Improving Web Application Security: Threats and Countermeasures*: Microsoft Press 2003.
- [10] A. Buecker, P. Ashley, M. Borrett *et al.*, *Understanding SOA Security Design and Implementation*: IBM Redbook Publication, 2007.
- [11] M. McIntosh, M. Gudgin, K. S. Morrison *et al.*, "Basic Security Profile Version 1.0," *WS-I Standard*, 2007.
- [12] J. Jürjens, "UMLsec: Extending UML for secure systems development," in *«UML» 2002 — The Unified Modeling Language*, pp. 412-425, Berlin / Heidelberg: Springer, 2002.
- [13] Y. Nakamura, M. Tatsubori, T. Imanura *et al.*, "Model-Driven Security Based on a Web Services Security Architecture." *International Conference on Services Computing 2005 (SCC'05)*. Florida, USA (2005)
- [14] C. Wolter, M. Menzel, A. Schaad *et al.*, "Model-driven business process security requirement specification," *Journal on System Architect: (Special Issue on Secure SOA)* October 26 2008
- [15] M. Hafner, and R. Breu, *Security Engineering for Service-Oriented Architectures*: Springer-Verlag New York Inc, 2008.
- [16] K. Bhargavan, C. Fournet, A. D. Gordon *et al.*, "TulaFale: A security tool for web services," in *Formal Methods for Components and Objects*, pp. 197-222, Berlin/Heidelberg: Springer, 2004.
- [17] T. Phan, J. Han, J.-G. Schneider *et al.*, "Quality-Driven Business Policy Specification and Refinement for Service-Oriented Systems." pp. 5-21. *International Conference on Service Oriented Computing 2008 (ICSOC '08)*. Springer, Sydney, Australia (December 2008)
- [18] T. Phan, J. Han, J.-G. Schneider *et al.*, "Policy-based service registration and discovery," *International Conference on Cooperative Information Systems 2007 (CoopIS '07)*. Villamoura, Algarve, Portugal (2007). *LNCS 4804*. pp. 417-427.