A Security Meta-Language for SOAP Messaging

ROBERT BAIRD
and
ROSE GAMBLE
University of Tulsa

Due to the increasing availability of competing service providers and the decreasing costs of moving services online recent trends in information systems development direct focus towards leveraging complex distributed system interconnections. To that end service-oriented architectures and web services have become commonplace in business and government application development because they facilitate rapid development and deployment through the use of standards that document interfaces and the message exchanges. However, the hierarchically related standards have complex documented interconnections and dependencies. The configuration of the services and the messages they exchange must adhere to the mandates established in these documents, yet the guidance offered by each specification is often too expansive for software developers to understand without assistance. Incorrect configurations can lead to messaging configurations that result in software vulnerabilities, system unavailability, service disruption, and ultimately loss of protected information. In this paper, we devise a Security Meta Language for secure web service communication based on a dynamic modeling framework. The framework models expert knowledge gathered from the intensive analysis of message protection protocols specified in web service standards. We outline a process to create and modify secure messaging directives through a case study investigating X.509 PKI tokens and digital signatures for SOAP communication.

Keywords: Security, messaging, SOAP, web services

1. INTRODUCTION

Service-oriented architectures (SOA) are a popular choice for systems development given the now mature concepts of service discovery, resource sharing, always-on service availability, and cloud computing. The interconnection of complex distributed systems has increased the availability of competing service providers and decreased the costs of moving services online, exploiting web services to promote rapid development of these integrated systems. Web services are built on standardized interface specifications to facilitate their integration and efficient execution. Businesses and governments use these specifications to find, evaluate, select, and integrate services, combining internal and external services to form their SOAs.

Communication between service endpoints frequently occurs over un-trusted networks meaning that end-to-end message security is unreliable or impossible given certain circumstances. Traditional mechanisms to secure communication rely on transport layer protection mechanisms such as SSL and HTTPS. Web services communicate through HTTP requests and complex invocation patterns, including task delegation, and delayed responses can cause the existing transport layer protection mechanisms to be insufficient to meet security needs. In response, significant message protection mechanisms have been created and codified as *web service standards* [W3C, 2012].

These standards, collectively referred to as WS-*, enhance the basic messaging of web services through extensions to their underlying core communication platform. One core WS-* standard is the SOAP messaging protocol, a base framework enabling web services to store application related data in a message payload and isolate security related extensions into a message header [W3C, 2007]. The message header is extensible, meaning that new standards can institute added features, and features can be mixed together to provide different configurations of security. The number of extensions, and their complex inter-referencing, make their correct and proper instantiation problematic. A tremendous amount of information must be reviewed and checked to ensure proper message configuration. Incorrect specifications can lead to disastrous application configurations

leading to software vulnerabilities, system unavailability and service disruption, and ultimately loss of secure protected information.

In this paper, we introduce a security meta-language (SML) that directs the configuration of secure SOAP messages to reduce the burden associated with ensuring proper specifications. We define a dynamic model to specify security directives regarding organizational expectations for secure message communication. Our model builds on earlier work in which we modeled the interrelationships of WS-* standards [Baird and Gamble, 2011]. The Security Meta Language we express in this paper provides end-users with a novel language to represent the stringent standards-based requirements required by different application scenarios (e.g. specific security policies) with the fluidity of web services (e.g. unknown SOAP message payloads and security token values). The language provides two major contributions for SOAs: (1) a mechanism to explicitly visualize the configuration of secure SOAP messaging, and (2) a framework for WS-* dependencies in secure messaging to be understood, justified, and adjusted for different architectures. As such the SML is a dynamic method for adapting security directives as application needs change or errors in published messaging standards are manifested.

In the next sections, we review research related to a general investigation of web services, modeling concepts, and security profiles in the Unified Modeling Language (UML), which we use to define our framework. We overview the static portion of our security meta-model framework [Baird, 2011], which includes the UML modeling artifacts we define to notate the WS-* standards. We then define the dynamic aspect of the framework that specifies directives for secure message configurations, producing the SML. Finally, we use the framework and SML to demonstrate attaching different security token types to messages, addressing different security control requirements.

2. RELEVANT WORK

This work leverages the benefits found in Web Services, their standards, the Unified Modeling Language, and other related research in security control modeling, which we review in this section.

2.1 Web Services

Service-oriented architectures with web services have become commonplace in business and government application development. Differing types of web enabled online services have evolved over time including web applications, software as a service, and cloud computing. In this paper, we primarily focus on traditional web services, that is, software functionality deployed onto an application server framework that abstracts internal functionality behind a standardized interface. These services have been researched from various security perspectives, including those involving service-oriented architectures [Rahman and Schaad, 2007], [Sitaraman, 2010], mobile applications [Ahmed, et al., 2014] and service clouds [She, et al., 2010], [She, et al., 2011]. The Web Services Architecture [W3C, 2004] outlines a set of service characteristics that enable these complex functionalities to exist including deployment descriptors and messaging protocols as performed by government entities and commercial industries. Thus, web services exist for a multitude of domains including energy trading, healthcare systems, e-commerce, telecommunications, banking, and scientific research systems.

A variety of organizations have collectively authored standards documents to define web services, the messages they exchange, and the protocols for their complex invocations addressing issues of security, transactions, service-level agreements, and discovery. These organizations include the World Wide Web Consortium (W3C), the Organization for the Advancement of Structured Information Standards (OASIS), the Object Management Group (OMG), xmlsoap.org, IBM, Microsoft, and Oracle, to name a few examples. Though web services may use different communication patterns, the de facto mechanism through which web services communicate is the Simple Object Access Protocol (SOAP) WS standard [W3C, 2007]. We visualize the connectivity of the web services, clients, the standards organizations, and messages in Figure 1. Clients and web services communicate with each other by crafting Extensible Markup Language (XML) mes-

sages that are requests and responses. Each message must conform to a set of standards drafted by the agreed upon organizations.

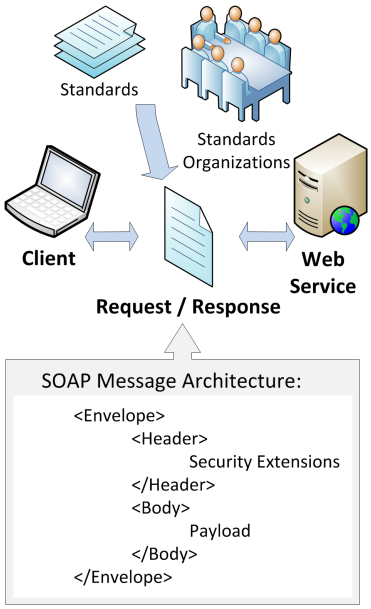

Figure 1: Web Service Messaging Architecture.

Frequently the standards are expressed in-part with XML [W3C, 2008], a hierarchical language with elements and defined data structures. There are several different communication patterns that are available for configuring a web service architecture [W3C, 2004] including request/response, conversation, delayed-response, and one-way. SOAP establishes a mechanism for the web services and clients to refer to object data and encode it in XML to communicate over the Internet. Other protocols, such as REST and JMS for example, are outside the scope of this research. However, any XML-based communication protocol can benefit from the framework described in this paper since it resolves many issues regarding the complexity for configuring secure web service introduced by employing WS-* standards.

Shown in Figure 1, the SOAP message header is a portion of each web service communication that allows for different security extensions to be expressed and attached. We refer in this text to the different extensions as *security controls* due to the target of our Security Meta Language. These security controls are more detailed and message-specific than those found in the NIST SP800-53 for federal information systems [NIST, 2013]. One such example of a security control is the application of a digital signature to the message. The problems with using SOAP messaging given the differences among desired controls stem from the extensible nature of XML and the complexity of the WS-* standards. In previous research ([Baird and Gamble, 2010], [Baird and Gamble, 2011]), we outlined some of the difficulties of integrating WS-* standards into service oriented architectures [Carlo, Albers, and Hao, 2006], [Hepner, Gamble, and Gamble, 2006], [Bhargavan, et al., 2004]. In summary:

—a multitude of different standards exist, all of which must be reviewed by an expert architect to understand their applicability and interrelationships,

—each standard frequently defines multiple security controls within a single document, for example WS-Security defines a variety of different security token types each for different needs, and

—many standards lack concrete directives for universal applicability, as is the case with XML-Encryption, which can be used to encrypt different portions of a SOAP message.

The requirements of a software architect or security engineer to understand the connectivity among and between the different standards is quite exhaustive for a correct and complete integration of secure SOAP messaging. In many cases it is simpler to say a toolkit supports a specific standard, and that a standard integrates the use of said toolkit, but this support may not be coupled with verification. A more exhaustive mechanism is needed to check and understand that the messages each web service exchanges is both syntactically and semantically correct with regards to the application of security controls. Some existing research in this area of XML investigation does exist. Specifically, the TulaFale language [Bhargavan, et al., 2004] targets the XML WS-Policy statements of web services using pi calculus to verify the code generates SOAP messages with the desired properties as stated in each policy document. A disadvantage of this approach is the limited support for the different WS-* standards. Furthermore, an intermediary script language must be learned by any user of the system, increasing complexity while reducing the general level of understanding that a software developer has about what the detailed configurations of the WS-* standards intend.

2.2 Unified Modeling Language

The Unified Modeling Language (UML) is a multi-level, object-oriented modeling framework providing a specification language for real-world objects built on top of a higher-level metaclass framework. UML has structures for describing object-oriented concepts of classes with attributes and operations (via class diagrams) as well as other complex structures including use cases and state charts. UML allows us to model the messages that web services use for communication. As such UML is a natural choice to model web services and SOAs due to the ease with which components can be diagrammed, the interfaces can be modeled, and their interactions can be explicitly sequenced.

Although UML models can be stored in an encoded XML notation [Object Management Group, 2007], UML itself does not explicitly include the necessary framework to model the descriptive documents of XML specifications (schemas). UML is built upon a multi-level modeling paradigm that specifies core language features including entities, relationships, attributes, and names. The core language is then refined to produce usable modeling artifacts, such as a *class*, that a software developer can use in a model or diagram. Generating a new diagram uses artifacts from the meta-class structure. A software developer can then model real-world objects by creating class diagram objects inside a UML diagram.

Since UML is extensible, it enables developers to extend core modeling functionality by expressing new artifacts known as *stereotypes*. Multiple stereotypes are collected into a UML *profile* for easy documentation and distribution among software developers. In UML, a stereotype is defined by extending (through a concept of generalization) from the core UML meta-class to refine and create new modeling constructs. Standard UML modeling practices dictate that using a stereotype to describe a class is accomplished through a lower case text annotation of the stereotype name surrounded in gullimets (e.g. <<stereotypeName>>). That is, defining a new UML stereotype is accomplished by a UML Profile specification of linked generalizations, and using the stereotype is done through tagging it with the appropriate stereotype name.

UML Profile specifications are quite common within the UML modeling community as a result of their extensive nature and capacity to model complex domains. Such profiles have been authored for Service Oriented Architectures [Object Management Group, 2009], CORBA Component Models [Object Management Group 2011], and Enterprise Application Integration [Object Management Group, 2004]. We define the Security Meta Language (SML) using a new UML Profile specification. The SML builds on top of an existing static model we have defined [Baird and Gamble, 2011], which in part relies on the modeling of an XML Schemas specification developed by Carlson [Carlson, 2008]. We review both of these works in detail in the next section.

2.3 XML Schema Profile

Carlson [Carlson, 2008] developed a UML profile specifically for modeling general XML schemas. The profile specifies extensions to common XML Schema constructs for data types. SOAP messages and the WS-* standards are frequently defined through the specification of these data types. These include XML simple types, complex types, enumerations, sequences, and attributes. Table 1 serves as a mapping between the targeted schema element types we use in this paper.

Table I: XML Profile Stereotype names and corresponding XML Elements

Stereotype Name	Metaclass Inheritance	Corresponding XML
choice	Class	xsd:choice
sequence	Class	xsd:sequence
complexType	Class, DataType	xsd:sequence
simpleTyple	DataType	xsd:simpleType
any	Property	xsd:any
attribute	Property	xsd:attribute
content	Property	xsd:simpleContent
element	Property	xsd:element

Because WS-* specifications use XML Schema to define their data types and messaging configurations, the XML Schema profile is a natural choice to incorporate in a modeling framework for WS-* standards. However, Carlson's profile has major limiting factors: (1) an inability to model multiple Schema files simultaneously, (2) a lack of support for modeling security related concerns, and (3) the inability to track cross-document dependency links. In [Baird and Gamble, 2011], we extend the functionality of the Carlson UML profile by expanding the number of standards the framework supports, and by more rigidly defining the process through which users interact with the WS-SMF.

Overall, our WS-SMF models highly complex WS-* specifications because we deliberately structured it to represent each XML Schema in exhaustive detail. We review the relevant portions of our static model later in this paper. By retaining a link to the WS-* standard Schema specification our Security Meta Language (SML) remains standards-centric, and relevant to how the standards are actually used in real-world systems. A complete model of the WS-* standards for SOAP, XML-Encryption, XML-Signature, WS-Trust, and WS-SecureConversation has been specified using the modified-Carlson model in [Baird, 2011]. The model tracks cross-specification links based on the documentation of the WS-* standard and their intended use.

2.4 Security Modeling

In this section, we review related security modeling efforts that target or use different WS-* standards specification. Meta-modeling within WS architectures traditionally concerns the generation of semantic information to attach to existing WS standards documents. A variety of current standards can accomplish the level of annotation that services require. Work in [Lautenbacher and Bauer, 2007] establishes a meta-model across the different WS-* semantic standards (OWL-S, WSDL-S, WSMO, SWSF, etc.) to join different conceptual semantic models. Although these standards are XML in nature, the model targets the higher-level concepts across the different languages. Modeling elements and associations are presented in UML; however, no extensions to the base UML meta-model are created. Other approaches to model the different types of semantic information that can be attached to WS include examining the different technology layers associated with services and the types of standards that can be applied across them [Thuraisingham, 2005].

Researchers examining SOA security issues have focused on mitigation strategies, such as determining the existence of a violation by employing models to understand how assets traverse services [Menzel and Meinel, 2009]. Security patterns match detected problems with mitigation

design strategies. Complex security problems may require applying multiple security patterns. Dong and Akl [2007] use a model checker to examine composed security patterns for accuracy. Their model checker determines if the patterns retain core mitigation properties once composed. To apply the patterns, they interpret the architecture requirements surrounding the potential security problem. They translate these requirements to UML and compare them with composed patterns for consistency. The difference is that there is no well-defined method to get from the security control to security problem. The Service Matchmaker [Zheng-qiu, et al., 2009] uses an ontology-based approach to extend the WS-Policy standards document with semantics. The goal is to determine if two policy documents match. The claim is that without semantics, this matching produces limited results. The authors do not extend the semantics or address the inter-referencing problem of the standards documents that have complementary information.

3. STATIC MODEL

In this section, we review the WS-* Security Meta-model Framework (WS-SMF) and the static model it provides [Baird and Gamble, 2011]. To create the Security Meta-Language we leverage the existing static portions of the WS-SMF and add in a new dynamic model as shown in Figure 2. The static model provides a stable, reusable foundation on which the more complex layers can be formulated to accommodate the security directives we require.

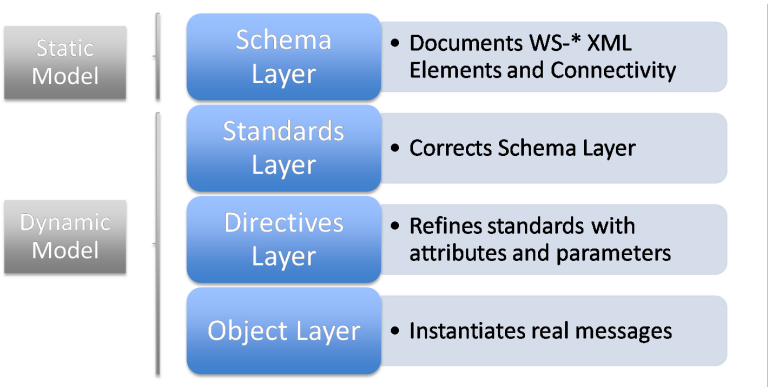

Figure 2: Multi-layered Approach.

The dynamic model relies on a multi-layered approach that links the static model with real world object instantiations of XML messages. The layers are the *Schema Layer* (static), *Standards Layer* (static), *Directives Layer* (dynamic), and the *Object Layer* (dynamic). The core contribution of the work presented in this paper is the *Directives Layer*. It serves as an intermediary between the static model and real world XML messages. For context, we review the existing *Schema Layer* [Baird and Gamble, 2011], [Baird, 2011].

The *Schema Layer* codifies different UML classes that are representative of the XML Schemas for each of the WS-* standards. Our representation uses a modified version of the Carlson UML profile [Carlson, 2008] to iteratively model each WS-* standard, the Schema files that each standard contains, and all of the XML elements that are defined within them. These classes are generated from the WS-* standards for WS-Security, WS-Trust, XML-Encryption, and XML-Signature, along with other standards such as WS-Trust and WS-SecureConversation. Because the model forming the Schema Later is quite exhaustive, we diagram partial views that target specific XML elements of interest. For example, we depict some of the different token types taken from the static model, shown in Figure 3.

Figure 3 shows the corresponding UML representations of different XML security tokens. Each XML element is shown with a corresponding stereotyped header, in this case <<complexType>> to refer to the corresponding type of XML Schema element, a name with included package

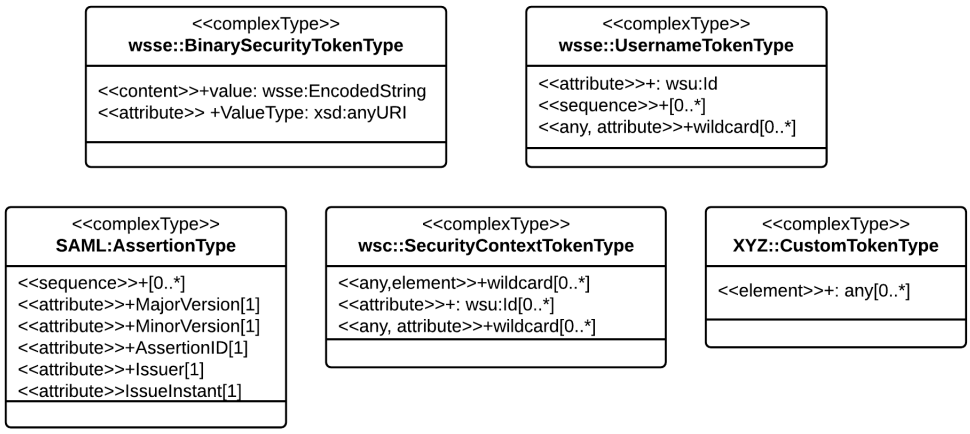

Figure 3: Variety in WS-* Tokens.

hierarchy (e.g., **wsse:UsernameTokenType**), and a set of corresponding XML sub-elements and attributes. Figure 3 shows XML elements from the different known standards, as well as a custom token type that is given the package name XYZ. The static model is stable in its architecture configuration, yet supports the ability to add new elements when new WS-* standards are drafted and used in service architectures. It is important to note that this model is not limited to the token types shown in Figure 3, although one of the benefits of using a static model is that we have identified several that are primary useful for security control modeling [Baird, 2011].

Within the domain of the WS-* standards there is a significant amount of flexibility and re-use designed into each Schema specification. For example, the **BinarySecurityTokenType** shown in Figure 3 can codify several different token types including X509 public key tokens, Kerberos tokens, and Rights Expression Language tokens [OASIS, 2012]. Each standard and XML elements must be used and carefully configured depending on the security control requirements of the message and application. The problem for many software developers is selecting the appropriate token and security control configuration for the needs of the application and deployment configuration. Not only must a standards specification user select the correct XML element among multiple competing standards, but once the selection has occurred, the element must be configured and integrated properly. Our modeling framework introduces new UML stereotypes to assist in resolving these problems.

In [Baird and Gamble, 2011] we introduced a cross-element dependency relationship framework to model the links among different WS-* standards documents. This approach uses an implementation-based understanding of the connectivity and use of the XML elements in a way that incorporates the re-use and integration that occurs among the different standards. Figure 4 shows such a view of the *Schema Layer* with associations that attach a WS-Security username token to a SOAP message header. In the view, <<direct>> associations are those for which the WS-* explicitly states a known relationship and <<elementof>> associations are links that occur through potential configurations through the use of several interacting standards documents.

The development of the static model uncovered several issues with respect to using the WS-* standards in any SOA environment that impact the design of the SML. Of primary interest is that different standards do not completely guide the correct configurations of SOAP messages. For example, the WS-Security standard that defines the **UsernameTokenType** element shown in Figure 4 does not explicitly designate the attachment of the token to the SOAP header. Instead the standard *suggests* that the token can be attached to a variety of different XML elements

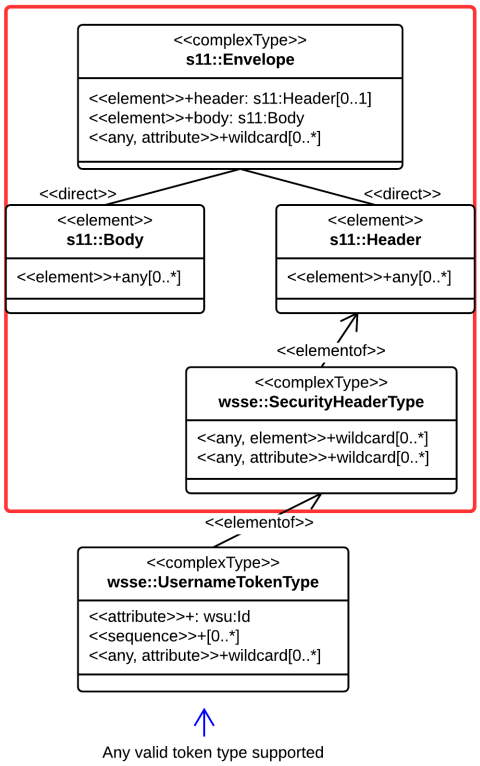

Figure 4: Schema Layer with Single Username Token.

to support reuse and flexible configurations. This variance burdens the software developer to correctly choose the elements and their attachment by navigating the different standards and their connectivity for specific security controls. Another issue found was that the XML Schemas for each standard often do not populate the XML elements with complete security relevant details. For example, in the case of the token in Figure 4, necessary sub-elements for username details, passwords, and timestamps are not specified at the XML Schema level. Instead, the WS-* standards incorporate the use of XML wildcards, that allow any XML element to fill certain sub-element relationships. To correctly determine the configuration of the SOAP message the software developer must reference both the XML Schema for type checking and the documentation of the WS-* to find example instantiations that document intended configuration of the standard.

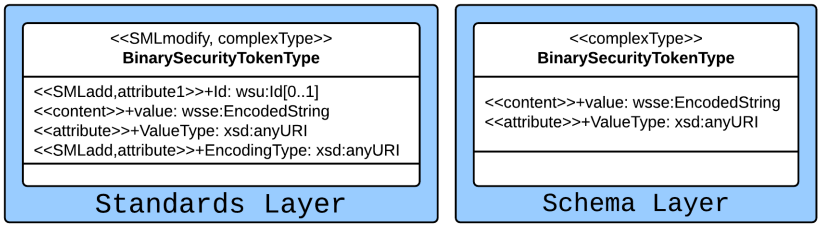

Figure 5: SML Modify.

Our investigation into the WS-* standards and the static model uncovered a number of XML Schema inconsistencies that must be addressed in any complete modeling framework. These inconsistencies are the result of mismatches between the WS-* documentation and the constructs that are provided in each standard′s XML Schema declaration. One clear example comes from a detailed examination of lines 759-796 of the WS-Security specification [OASIS, 2012] that shows attributes for the **BinarySecurityTokenType** element; yet the Schema does not outline these attributes. Ultimately, to instantiate a proper SOAP message certain attributes about token values are required and must be shown in the model. Our solution modifies the representative *Schema Layer* as shown in Figure 5. The *Standards Layer* is tasked with manipulating the view of the *Schema Layer* to introduce the necessary changes to guide SOAP message configuration. This layer introduces two stereotypes, <<SMLadd>> to mark the changes in attributes and elements, and <<SMLmodify>> to tag the updated Schema data type.

In Figure 5, we show the updated **BinarySecurityTokenType** along the left hand side in contrast to the existing artifact obtained from the *Schema Layer* and WS-* Schema files. The updated token includes new attributes for Id and EncodingType. A thorough examination of the WS-* standards reveals several of the modifications that must be taken into account when instantiating SOAP messages with security controls. This layer forms the transitional model entity between the static and dynamic models within the framework because it relies heavily on the static model artifacts but cannot adapt as much as to new directives as the remaining layers we present in the next section. However, it is a distinct enhancement that we isolate from the *Schema Layer* so that the changes can be tracked and understood for review at later dates.

Other problems with using the WS-* standards include ambiguity within specific token elements and a need for cross element and multiple message dependencies. For example, the Binary Security Token supports multiple token encoding types (e.g., X509, Kerberos, and REL tokens), a SOAP message may contain more than one reference to the same token or multiple token types, and often, the tokens are used within more complex security control structures, including digital signatures and encryption protocols. Our approach incorporates a dynamic partition within the framework that enhances modeling capabilities to address these issues and to generate the SML. We define this dynamic model by introducing parameters and directives, which we introduce in the following section.

4. SECURITY META LANGUAGE

In this section, we outline the dynamic modeling layers specifically related to the formation of the Security Meta Language (SML). These models enable the attachment of different security controls to messages through the expression of UML modeling constraints over the instantiation of SOAP messages. The SMLs inherent flexibility can incorporate the different modeling needs of a web service architecture as it changes over time, in contrast to the static nature of the WS-* documentation standards. The security controls or constraints depend on organizational expectations and requirements, as well as implementation specific details that reflect those expectations. Security requirements can originate from multiple sources including application needs, software developer concerns, or regulatory documents. We refer to the collective set of constraints that the web service architecture must adhere to as security *directives*.

4.1 Directives Layer

Directives are the codification of security controls attached to SOAP messages and are collected into a single *Directives Layer* within the framework to clearly isolate their separation from the existing static model and real-world XML objects. We refer to the collective set of directive specifications as *SML directives*. The framework outlines a set of SML directives that can be searched, examined, and used for to embed different security controls. To that end the SML directives in the WS-SMF forms a catalog of known security controls for secure SOAP messages. The *Directives Layer* defines stereotypes targeted towards the correct modeling of the SOAP messages. From a modeling perspective we construct a *Directives Layer* using the following basic

approach and modeling constructs:

—Determine which classes of the updated *Standards Layer* are targeted for the specific security control requested.

—Use the new <<SMLrefinement>> stereotype to generate a new class tasked with modeling the security control directive.

—Denote all refined classes with a <<SMLdirective>> stereotype in their header.

—Explicitly populate known attributes based on security controls and the WS-* standards guidance.

—Revise the modeled classes by introducing new parameters using the <<SMLparameter>> stereotype.

In the dynamic model, we use the *Directives Layer* as a definitive link between the abstract data types specified in the WS-* data structures and the real world object instantiations found in XML messages. Three key stereotypes accomplish this interconnection: <<SMLdirective>>, <<SMLparameter>>, and <<SMLrefinement>>. Table 2 overviews the new stereotype definitions that are part of the dynamic model, the specific UML metaclasses each generalizes, and a semantic description of the intent of the stereotype with any additional constraints the stereotype requires.

Table II: XML Profile Stereotype names and corresponding XML Elements

Stereotype Name	**Metaclass Inheritance**	**Semantics and Constraints**
SMLadd	Attribute	Corrects *Schema Layer* as necessary based on inaccuracies found in WS-* document analysis: (1) Models missing XML elements; (2) Models missing XML attributes.
SMLmodify	Class	Identifies and tags *Standards Layer* classes that have been modified using the SMLadd stereotype.
SMLdirective	Class, DataType	Used throughout the *Directives Layer* for class-level modeling of abstract security control relationships: (1) Names directive shared among several classifiers; (2) Removes unnecessary wildcards; (3) Names attributes and elements; (4) Allows parameters as necessary; (5) Directive classes are abstract.
SMLparameter	Attribute	Explicitly labels XML elements and attributes that cannot be modeled until *Object Layer* is instantiated.
SMLdependency	Association, Constraint	Models complex element and attribute dependencies between classes modeled in the *Directives Layer*: (1) Association links between two SMLdirective classes; (2) Constraints specified in Object constraint language.
SMLrefinement	Generalization	Intermediary link between *Standards* and *Directives Layers*: (1) Generalization arrow points from left to right; (2) Type from child to parent is retained; (3) Generalization permits the non-standard refinement from regular class (*Standards Layer*) to abstract class (*Directives Layer*); (4) Generalization removes directionality from associations between different classes within the *Directives Layer*.
SMLinstantiates	Generalization	Intermediary link between the *Directives* and *Object Layers*: (1) Generalization arrow points from left to right; (2) Type from child to parent is retained.

We introduce an example directive in Figure 6. On the right side of Figure 6 we display the necessary XML Schema classes selected from the *Standards Layer* to model the corresponding directive. Along the left side we place the <<SMLdirective>> stereotype in class headers that

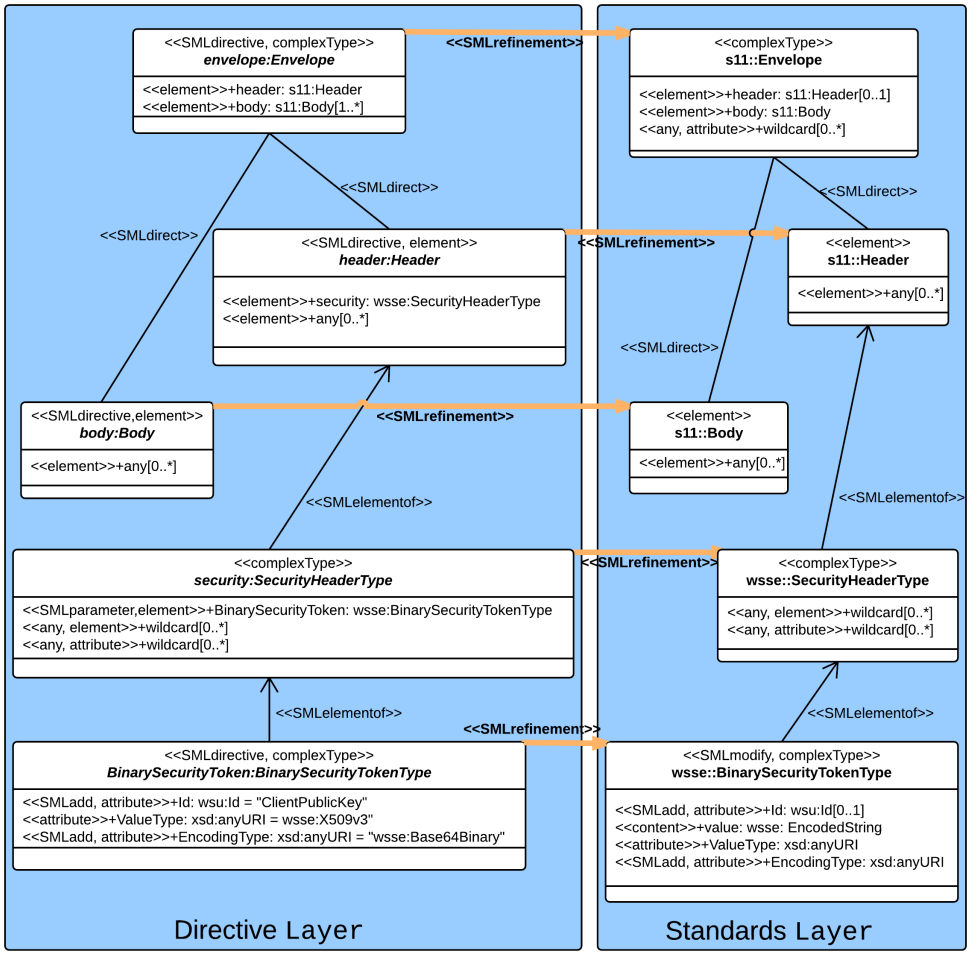

Figure 6: Directive Layer.

belong to specific security control attachments. There is a direct mapping from the left to the right for each class that is necessary to express the directive.

Figure 6 contains the classes that correspond to a directive to specify a SOAP message with an attached security token. This modeling approach involves the specification of the SOAP envelope, header, and body, as well as the Security header and token elements. We start our investigation into the mechanisms of the new stereotypes through an analysis of the SOAP envelope (located in the upper left hand side of Figure 6 with the text ***envelope:Envelope***). The class is marked with the <<SMLdirective>> stereotype that mandates the class is a named object (**envelope**), yet it is defined as an abstract class (marked via italic text). This mandate retains the concept that the SML directive is not yet a real SOAP message. Still the defined classes within the template must satisfy certain properties, such as the XML element attribute names, to embed the correct security controls.

The application of the <<SMLdirective>> stereotype to the **envelope** class allows some additional modeling constraints to be embedded within the directive. Specifically, the wildcard[0..*] attribute found in the corresponding **s11:Envelope** type is allowed to be removed from the directive because the directive for this SOAP message does not need it. This removal is not always necessary as shown in later classes for the SOAP body, for example; however, to simplify

the model unnecessary wildcards are removed. The copy and removal of different XML element and attributes are governed by the documentation of the WS-* standards and originates from a process of refinement that is modeled between the *Standards* and *Directives Layers*.

Each class defined in the*Directives Layers* must include a <<SMLrefinement>> stereotyped generalization pointing towards the *Standards Layer*. The refinement accomplishes three main things. First, the application of this stereotype relaxes the directionality link that exists between all relationships that emerged within the *Standards Layer* description. The intent of this model specification relaxation is to simplify the model by inferring it through the analysis of the named object in the*Directives Layers*. As shown in Figure 6 the children of the **envelope** class (body and header) are not linked with <<direct>> or <<elementof>> stereotypes. Analyzing the class names for each XML element can derive these links.

The second major accomplishment of the refinement generalization is the typing modification, such as that which is applied to the **envelope** class. Traditional UML modeling constructs would stipulate that a generalization of the **s11:Envelope** class defines a new and enhanced version of the *Standards Layer* class. This is distinct from the objective of the SML, since we do not need to generate modifications to this layer for this part of the model. In contrast, we adopt the visual arrow to indicate that the child class (in this case, the **envelope**) is of the same type as the parent class (in this case, **s11:Envelope**). This refinement facilitates data type checking as well as grounds the modeled directives in the static portion of the framework (WS-SMF). Finally, the generalization performs the non-standard refinement of a UML object of the data type in the *Standards Layer* to the abstract class required to model the directive as mentioned in the previously.

The definition of the header and body elements of the directive follow the general guidelines established in Table 2 and described above, with one minor exception. In each of these classes, the wildcard specifications for elements have been retained. We leave this in place to show the reader that the model can define partially specified directives in the absence of full knowledge about the execution environment. This allowance is very similar to the stereotype used in the security header class shown in Figure 6 for a named <<SMLparameter>>. This stereotype tags certain class attributes and elements as parameterized fields that require additional information prior to object instantiation. As shown in Figure 6, the **BinarySecurityToken** object is a tagged parameter since its value (e.g. the binary data) is unknown until the SOAP message is fully crafted. Attributes can require parameterization due to factors including runtime constraints (e.g. modeling SOAP messages yet not knowing the full invocation details until a later time), privacy concerns (e.g. a software developer not wishing to store secret password information in a UML model), and space issues (e.g. complex binary encodings of tokens requiring a large amount of visual modeling space). For example, in username token keys, the username and password do not need to be stored in the directive. Other examples of parameters include the key values of X509 digital tokens and the calculation of signature values that are frequently unknown until runtime, since digital signatures and encryption values are based on the message input, random variables, and the keys generating the signature values.

Not all attributes in a codified directive need to be tagged as SML parameters. As shown in Figure 6, for example, the **BinarySecurityTokenType** element has several named attributes that have specified values including the Id, ValueType, and EncodingType. These are all of the values that can be predicted before runtime and incorporated into the specification of the modeled directive.

4.2 Object Layer

The final layer of the dynamic model that established the SML is the *Object Layer* as shown in Figure 7. Recall that the existing Standards layer grounds the modeling framework in the XML Schema elements that the WS-* standards provide. The *Directives Layer* expresses specific constraints on how those XML classes are instantiated, yet it is incomplete due to certain parameter details that are unknown at the time the directive is expressed. The final *Object*

Layer takes the stated requirements and instantiates known parameters to create XML objects. The object instantiation relies on a new stereotype <<SMLinstantiates>> that connects to the abstract directive classes to real world objects. This stereotype operates very similarly to the <<SMLrefinement>> generalization defined in the previous section and in Table 2. In Figure 7, for example, the XML message includes a **BinarySecurityToken** filling in the existing parameter data for token data with ClientKeyData. Parameterized values and the SOAP messages that have these values populated for real SOAP messages within the dynamic model fulfill the final aspect of generating the complete SML. The language in this case illustrates the dynamic models capacity to specify security control directives for SOAP messages with a variety of different parameterized values. It also shows that the SML is not limited to single example case study messages.

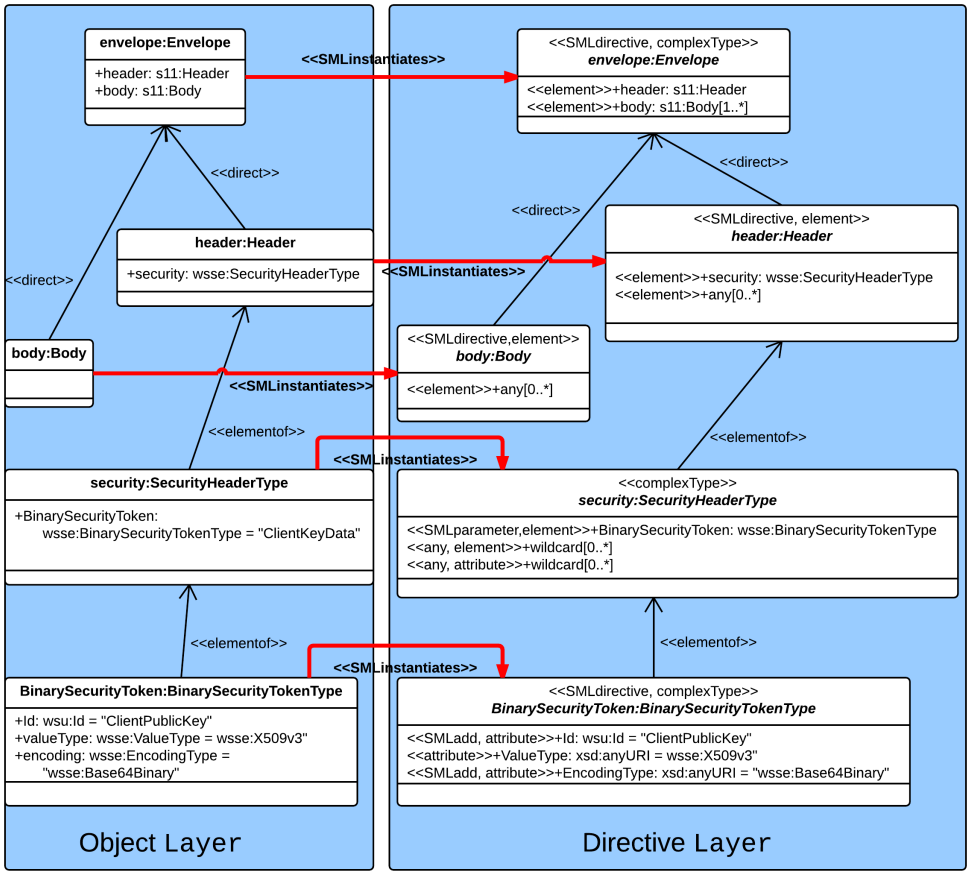

Figure 7: Object Layer.

Object Layer modeling is not a direct requirement for all SML directives; however, it serves as a mechanism to show that with certain input parameters we can codify real XML messages giving credence to the meta aspect of the model. Based on the information provided for the *Object Layer* of Figure 7, the dynamic model generates the XML document shown in Figure 8. The *Object Layer* provides the dynamic model with a simple and effective mechanism to check directives for accuracies. When real SOAP message objects are generated that fulfill the security control requirements, can be successfully type checked using XML namespace and data type

declarations, and when the SOAP messages can be used in real world SOA application settings, then the directive is known to be correct.

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..." >
     <S11:Header>
          ...
          <wsse:Security>
               <wsse: BinarySecurityTokenType>
                         <wsu:Id >ClientPublicKey</wsu:Id>
                         <wsse:ValueType>wsse:X509v3</wsse:ValueType>
                         <wsse:EncodingType>
                              wsse:Base64Binary
                         </wsse:EncodingType>

                              … Client Key Data …

               </wsse: BinarySecurityTokenType>
          </wsse:Security>
          ...
     </S11:Header>
     ...
</S11:Envelope>
```

Figure 8: XML Instantiation of Binary Security Token.

The process of taking the WS-* documentation, refining its view into usable UML classes, establishing constraints over their object instantiation, and finally generating real-world XML objects is the primary goal of the modeling framework. We define how to correctly identify the usable elements across the multiple WS-* standards and align their proper configuration given a base set of message requirements. In the following section we investigate a more in-depth case study dealing with XML digital signatures attached to SOAP messages, and how the SML directives support the specification of even more complex dependencies. We do so through the introduction of a reusable directive modeling process.

5. DIRECTIVE MODELING PROCESS - CASE STUDY

As stated previously, the SML contains a catalog of known directives for the instantiation of SOAP messages. It is not the goal of this research to outline all possible XML messages, or even all SML directives, but instead to outline a process for reusable directive specification based on application requirements. Under ideal circumstances a software developer would search the SML catalog for the correct directive and populate necessary SML parameters to generate a correct application configuration. An example scenario could include which web services are deployed in a government setting and that communication must be digitally signed using an X.509 public/private key pair. The SML directive in this case would be to apply the digital signature, the communication protocol would include SOAP messages, and the application developer would need an XML representation for the keys. In such a situation a software developer would search the SML catalog to find the desired message configuration. Due to the simplistic nature of this scenario it is highly unlikely such a basic directive would not exist. However, to illustrate the process for generating a new directive, we outline the steps to follow in these situations. In this section, we describe our modeling process to configure new directives for the SML. As an example we model the attachment of an X509 public key to a SOAP message with a digital signature. Figure 9 presents a brief outline of the four-part process for directive creation.

Figure 9: Directive Modeling Process.

When new SML directives require modeling, the software developer can follow the process in Figure 9. The process reflects situations that occur due to the release of a new software toolkit, introduction of new security control technologies, or a general shift in operational goals for an organization that requires the incorporation of new WS-* standards. When existing directives are already in use and need to be adapted to new needs, or corrected to fix existing problems, we outline a different process for directive modification in a later section. It is through the specification and definition of the SML directive stereotypes that the SML framework provides a specification language for reusable security controls applied to SOAP messages.

5.1 Control Selection

The first phase of the security modeling process, control selection, involves the selection of XML elements from the static model to achieve the desired security control. This phase takes input from the software developer and is heavily influenced by the application environment, configuration of web service endpoints, and the goal statements of the WS-* standards documents. Ultimately, control selection refers to the XML element identification at a class level in the UML model. The existing static model assists in this process by explicitly denoting which XML elements share sub-element relationships across and within the different WS-* standards.

For the case study, to apply a digital signature the software developer is sent to the SOAP and XML-Encryption standards given a reasonable base knowledge of the WS-* documents. The software developer must rely on the static mappings and links established in the *Standards Layer* to determine correct specifications, XML elements, and examples to review in this process. Through investigation, the **SignatureType** class is located within the *Standards Layer* of the dynamic model and is shown in Figure 10. The static model guides the software developer in understanding that this XML element is a core class; it has a fit within the XML SOAP message, and it has sub-element dependencies. Given the needs of the environment and application, the software developer must find the correct controls, even multiple security controls, to drive the dynamic process.

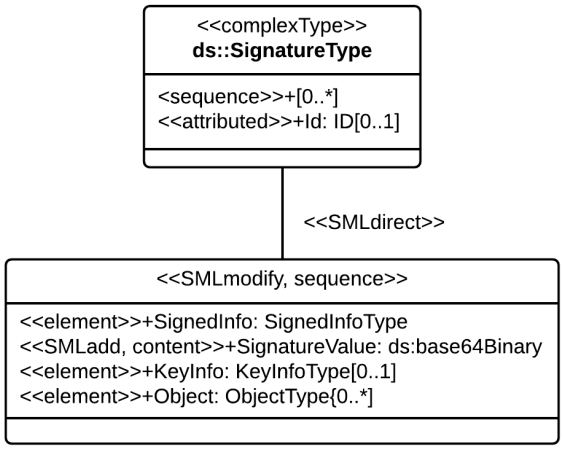

Figure 10: SignatureType (Standards Layer Partial View).

It is important to note that this phase of the dynamic modeling process does not involve making modifications to the static model or *Standards Layer*. These can be used as accepted practice. The existing modeled classes in the *Schema Layer* exist and are unchanged, even if they do not appear selected when the phase is completed. This is why the control selection presented in

Figure 10 is referred to as a partial view of the *Standards Layer*. Furthermore, the complete XML element selection is not required for this phase of the modeling process. Only the core security control elements, in this case the **SignatureType**, are required for the process to continue.

5.2 Message Attachment

Once the necessary security controls elements have been identified and located within the Standards layer according to the application requirements (e.g. message protection through signature, authentication, and encryption), the second phase of the modeling process is performed. During message attachment, the XML structures are navigated using the <<SMLdirect>> and <<SMLelementof>> associations to attain a path whereby the control is attached to the SOAP message envelope. Such a path for the **SignatureType** is shown below in Figure 11 and is guided by the WS-* documentation.

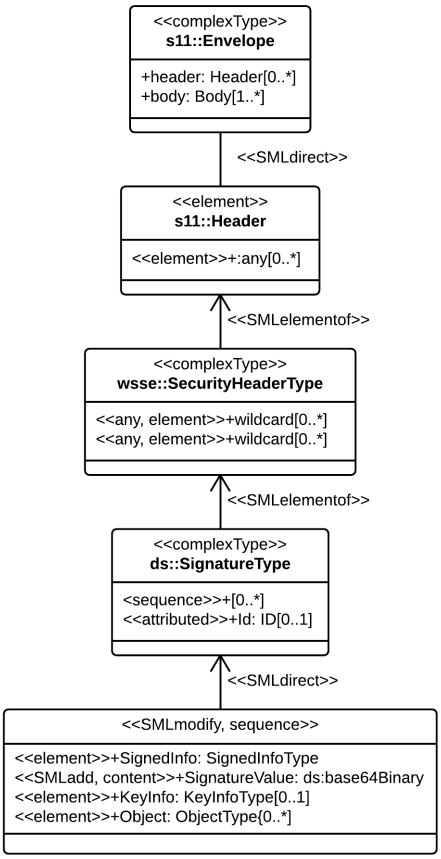

Figure 11: Element Trace for SignatureType (Standards Layer Partial View).

The correct path must be chosen according to the intent of the WS-* standards documentation. A digital signature, for example, can be embedded in any part of an XML document. However, for the correct application in this case study, which requires the attachment of the digital signature to a SOAP message, the **SignatureType** is a sub-element of the WS-Security **SecurityHeaderType**, which is a sub-element of the SOAP message Header, which is a direct sub-element of the SOAP Envelope. The path traversal process may, and frequently does, cross different WS-* standards to ultimately reach the SOAP Envelope. As shown in Figure 11 the

attachment links the XML element selected in the previous phase of the dynamic process. Once this is accomplished the next phase expands the number of identified XML elements outward to meet specific WS-* standard and messaging requirements.

5.3 Element Requirements

Upon locating the attachment path, the next phase of the dynamic modeling process involves crafting the SML directive in such a way that all element requirements are satisfied. This phase is the most exhaustive effort in the directive creation. Every required element in the Standards Layer for the targeted classes must be specified through a <<SMLdirective >> attribute. The specification of optional element attributes, such as the KeyInfo attribute of the **SignatureType**, is left to the interpretation of the WS-* standards documentation by the software developer.

Figure 12 introduces the directive for the digital signature, which due to its size is investigated in parts below. Figure 12 shows a decoupled view of the digital signature where only the *Directive Layer* is visible and a partial set of classes is arranged hierarchically to assist with visualizing the nested XML element structure. The tree structure is crucial for the software developer to understand how the SML directive is crafted. The author of the SML directive must take care to know that each element in the tree structure ensures that the attributes (sub-elements) are correctly specified.

To fully declare an SML directive, it is broken down into key sub-specifications. These are the digital signature element itself (**signature:SignatureType**), data and attributes regarding how the signature is calculated (**signedInfo: ds:SignedInfoType**), the value of the signature when it is calculated (**signatureValue: ds:SignatureValue**), and a reference to the key that is used to perform the signature (**keyInfo: ds:KeyInfo**). We notate two items of importance regarding this directive. First, the signature value is not known within the specification of the directive due to a lack of important information necessary for its calculation (i.e., the message payload and the key data). Thus, it is stereotyped as an <<SMLparameter>>. Secondly, the digital signature requires an X509 security token for the **KeyInfoType**. To that end, the developer must add the BinarySecurityToken directive to the SOAP message security header. This addition is based on the application configuration, the fact that the web services encode their security tokens in specific ways, and the fact that those codifications require the WS-Security element classes from the static model.

Completing the full element requirements phase of the dynamic modeling process requires performing an iterative tree-traversal to ensure all element attributes are specified in accordance with the WS-* standards and the requirements of the application. This portion of the process relies on the use of the <<SMLrefinement>> and <<SMLdirective>> stereotypes to both populate the correct classes and express constraints on the attributes to meet security concerns. Figure 13 provides a closer view of the relationship between core XML Signature elements from the *Standards Layer* and the SML directive specification. For example, the signedInfo directive fulfills all element requirements through the sub-specification of the canonicalizationMethod, signatureMethod, and reference objects. These requirements stem from the <<sequence>> defined for the SignedInfoType. Similarly, the directive for the canonicalizationMethod object contains a single attribute for the Algorithm, with an appropriate algorithm value set as "...xml-exc-c14n". The algorithm attribute is taken from the XML-Signature standard documentation and refers to the Exclusive XML Canonicalization algorithm as defined in [W3C, 2001]. The directive for the signatureMethod object documents an example of an optional attribute field (HMACOutputLengthType) which is not necessary for the model specification and is subsequently dropped from the attribute list. This deletion is allowed by the <<SMlLrefinement>> stereotype since not all attributes are necessary to satisfy given specifications.

The last attribute of the signedInfo object is the **reference:ReferenceType** object that defines the object over which the digital signature calculation is to be performed. In this case, the software developer understands the requirement for the digital signature must be expressed over an explicit XML element. In the example, this is the entire SOAP message body. Also,

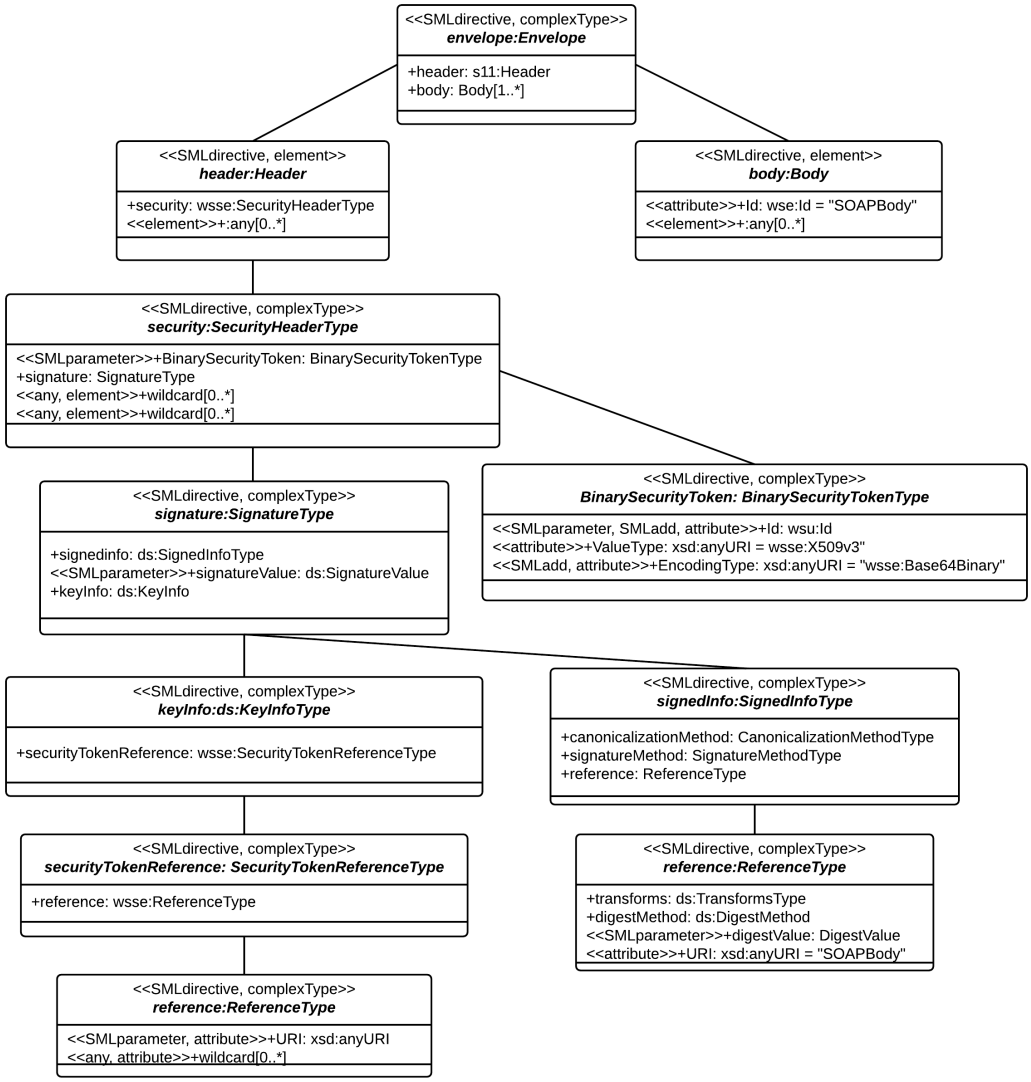

Figure 12: Directive for Digital Signature (Directive Layer).

the URI attribute is given a tagged name of SOAPBody. Other attribute fields (transforms and digestMethod) require the creation of other sub-element directive classes.

Once all sub-elements have been fully instantiated in the final phase of the dynamic modeling process, the SML directive must express the logical dependency requirements to exist between attributes of the SML classes in the *Directive Layer*. This phase enables the software developer to codify XML element and attribute interrelationships found to exist within the WS-* documentation. Dependency modeling is accomplished through the use of a UML stereotype <<SMLattributedependency>>. We show the dependency modeling process as part of the next section, illustrating how the directive can be adapted within the dynamic framework. Generally dependency modeling would occur immediately during the creation of the directive.

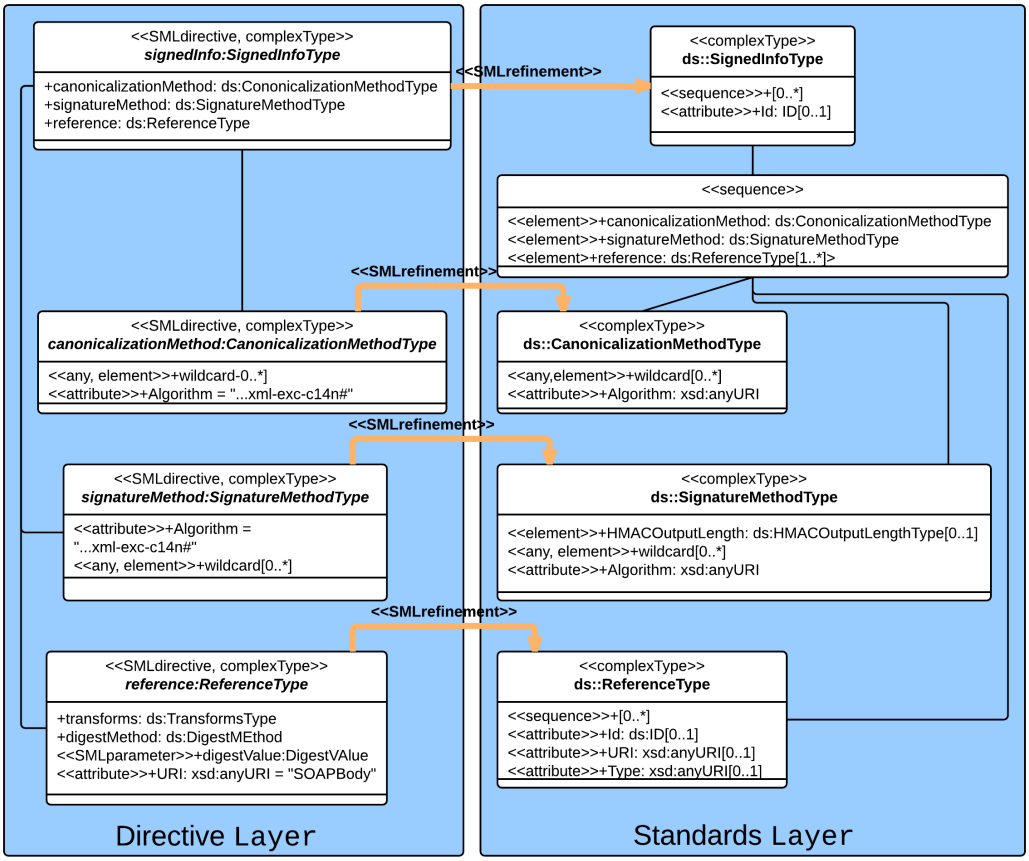

Figure 13: SignedInfo Attribute Specification.

6. DIRECTIVE RE-USE AND RE-SPECIFICATION

Dependency modeling is the final aspect of directive specification. However, as is often the case with WS-* use, there are problems with respect to correctly integrating and selecting the necessary message configurations. The ability for our dynamic model to adapt over time is an important property that is necessary to resolve these issues. In many cases the dependencies within the different WS-* standards and XML elements are explicitly known at the time a directive is created. For example, in the case of using any type of token and digital signature there will be a requirement that specific ID attributes match throughout the entire message. In the case of the specification of more complex relationships with multi-part message exchange, these dependencies are often only inferred through software developer knowledge as time goes by and a software product evolves. From this reasoning, we constructed the *Directive Layer* to be flexible with respect to modifications. As changes need to be made to directives, the users of the dynamic model can browse, select, and adjust the directives as necessary. In this section, we show how the existing directive for the digital signature can be altered when it is discovered that changes are needed to achieve correctness.

Figure 14 shows an updated view of the existing directive for a digital signature using an X509 public key token. Two different types of overlay boxes have been introduced over different sections of the message. After review of the directive, the SOAP messaging standard, and the digital signature two key dependency issues are manifested. First, shown in Figure 14 with a solid

line, the referenced token used by the digital signature must match the attached token included in the header of the message. Secondly, shown in Figure 14 with a purple line, the referenced digital signature must point to the ID element of the body of the SOAP message. Different tokens, digital signatures, and message configurations would each have unique corresponding dependency requirements. However, a user of the directive would note that the directive as specified in Figure 14 lacks any of the correct model dependencies.

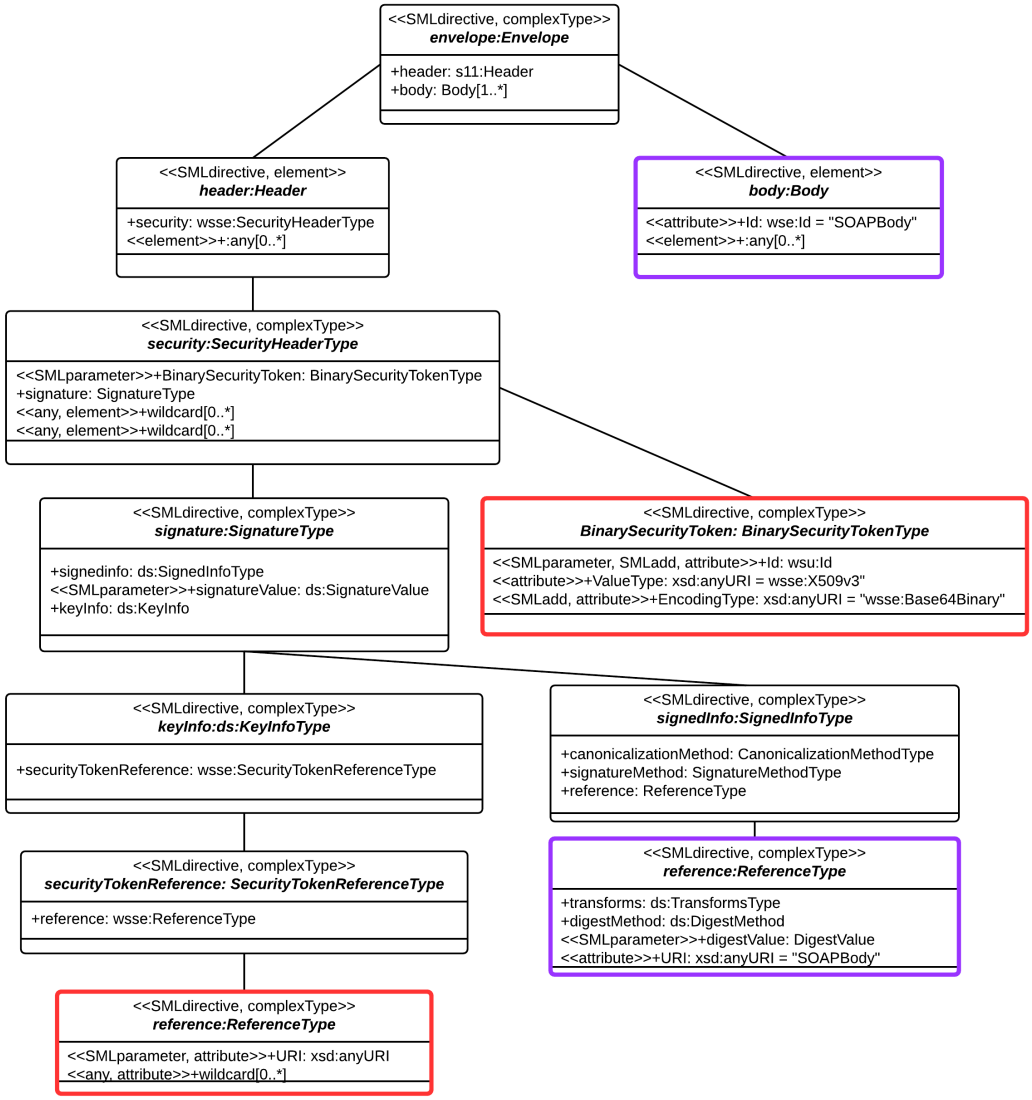

Figure 14: *XML-Security Digital Signature* Directive.

To correct this problem the directive must be updated to add the necessary constraints. The dynamic model includes a <<SMLattributedependency>> stereotype that adds the final structure necessary for comprehensive modeling of the complex SOAP message exchanges present in the web services architectures. Figure 15 shows an example dependency for the token ID requirement. Each dependency links two XML element classes and contains a constraint. Constraints

are expressed using the Object Constraint Language (OCL) [Object Management Group, 2010] and Boolean based on elements and their attributes as defined in UML.

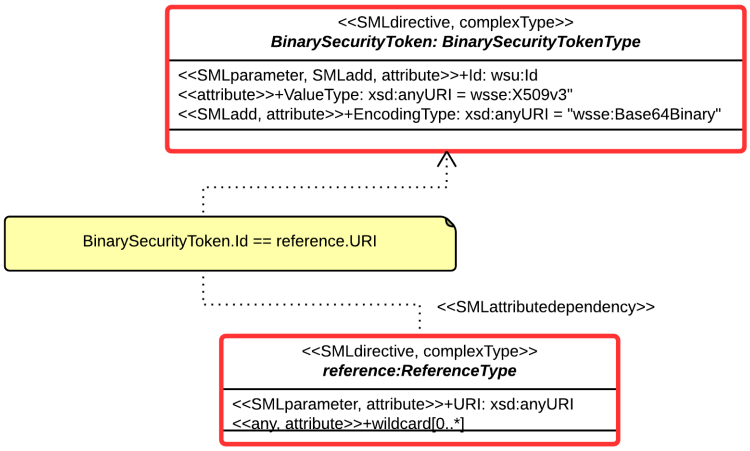

Figure 15: Digital Signature Dependency.

The most common use of the <<SMLattributedependency>> stereotype is the expression of == indicating the left hand side expression must match the right hand side expression; however, any valid OCL statement can be used. In Figure 15, the constraint states that each ID reference must match the other. Since these values are tagged as <<SMLparameter>> attributes, this type of dependency cannot be verified until runtime or a corresponding *Object Layer* model is created. The dependency for the SOAP body element is shown in Figure 16.

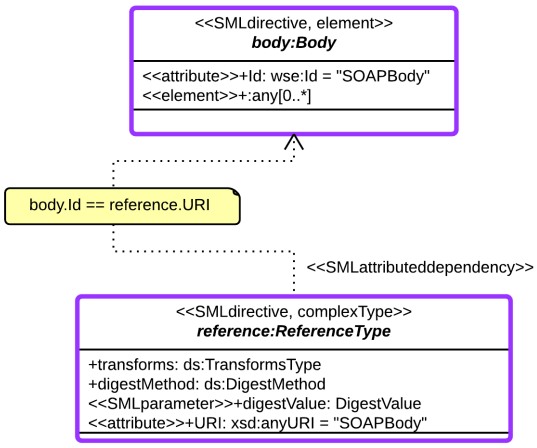

Figure 16: SOAP Body Dependency.

In this case, both the URI and ID attributes must equal SOAPBody in order for the digital signature to exist over the SOAP body element. From the directive specification and attribute values it is easy to view that this dependency is already satisfied. The updated directive that incorporates each of the dependency specifications is shown in Figure 17.

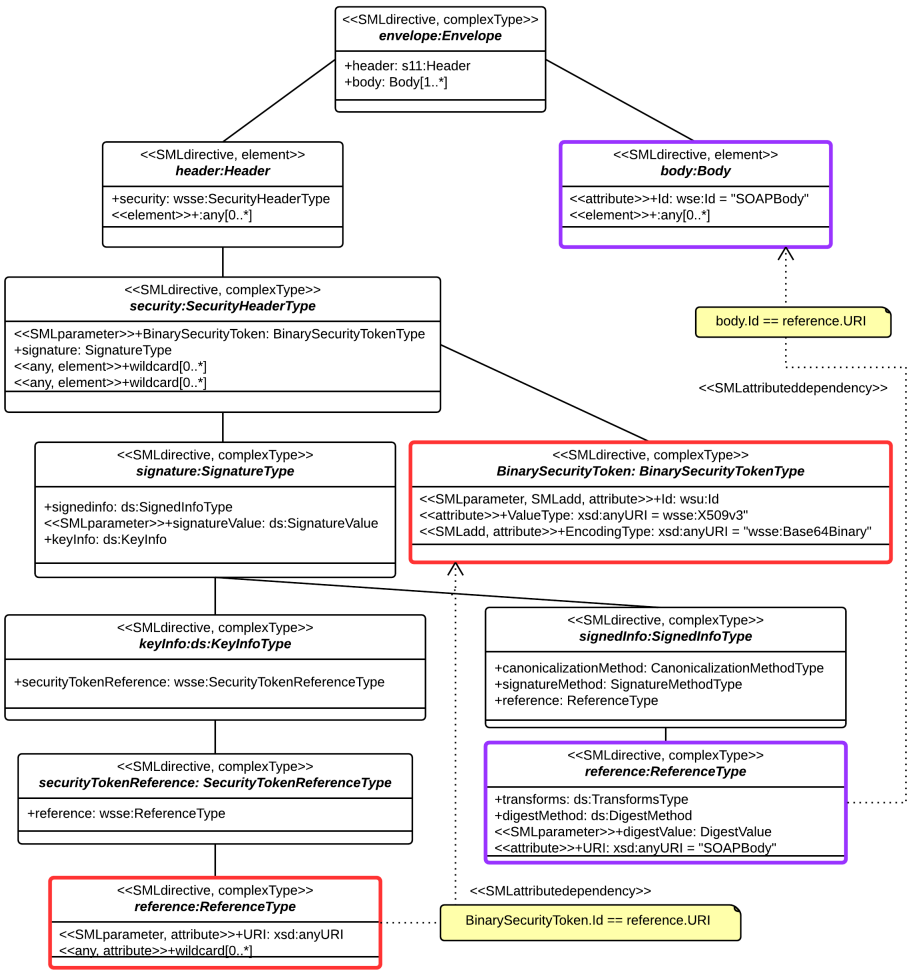

Figure 17: Updated *XML-Security Digital Signature* Directive.

We define dependency directionality (e.g. the establishment of child/parent in the relationship) based on a sub-element analysis and message ordering. For example, if expressing a single XML document, the element dependency denotes the child in the relationship as an XML element that is nested further down in the XML hierarchy. In multi-message exchanges (such as with WS-Trust RST/RSTR exchanges), the directionality is established such that the child element is associated with the message that is generated first in the sequence.

Fully specifying all dependency requirements is the final task in the dynamic modeling process outlined in Figure 9. Any object instantiation that the software developer codifies, generating SOAP XML messages, must satisfy the SML directive and dependencies. Users of the SML only need to invoke the modeling process when the constraints of the application environment specify controls and configurations that have not been modeled previously. In the previous example the existing specified directive was selected and updated to correct it and align the specification with the established WS-* standards. An alternative method of update to the directive is also available in the dynamic framework wherein the existing directive is selected and modified to perform a new functionality. We do not review this process in this work; however it is clear how the digital signature case study we present can be modified to use a variety of different token types from Figure 3, or different security control structures found within the static model. In

these cases the directive is copied, adjusted, and added to the existing catalog of known directive statements.

7. CONCLUSION

In this paper we define a Security Meta Language (SML) as a two-part model and dynamic process that documents the security relevant portions of the standards for their consistent, comprehensive, and correct application. We leverage an existing static model build in UML and based on XML Schema specification by adding to it the stereotypes necessary to model security control directives for web services. Figure 18 provides a complete view of the interconnected layers of the SML showing a portion of the complete language.

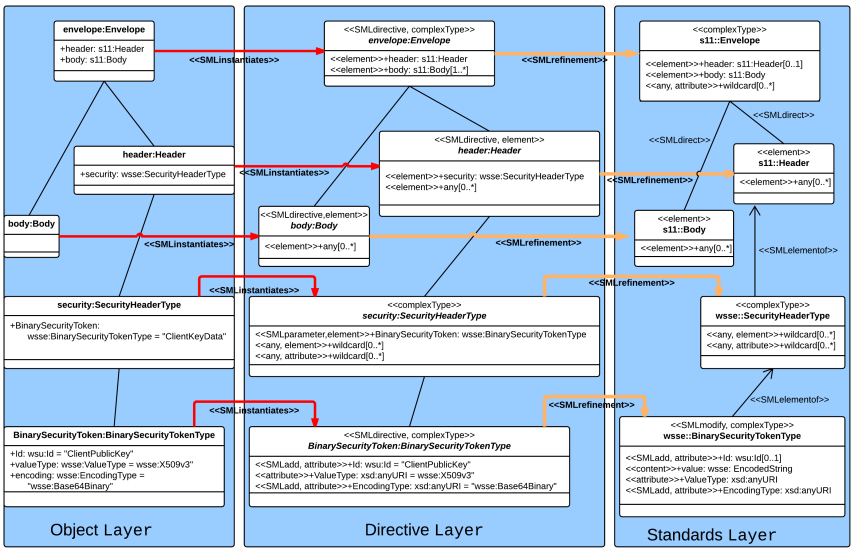

Figure 18: Sample Directive of the SML - SOAP Username Token.

REFERENCES

W3C 2012. Available from *http* : *//www.w3.org/standards/webofservices/*.

W3C 2007. *SOAP Version 1.2 Part 0: Primer (Second Edition).* Available from *http* : *//www.w3.org/TR/soap*12 − *part*0/.

Baird, R., and Gamble, R. 2011. Developing a Security Meta-Language Framework. In *Proceedings of the Hawaii International Conference on System Sciences.*

Baird, R. 2011. A Security Meta Language for Using Web Services Security Standards. Ph.D. Dissertation, University of Tulsa, Tulsa, OK.

Rahaman, M.A., and Schaad, A. 2011. Developing a Security Meta-Language Framework. In *Proceedings of the Hawaii International Conference on System Sciences.*

Sitaraman, L. 2010. Interoperable Security Standards for Web Services. In *IT Professional*, Vol. 12, No. 5, pp. 42-47.

Ahmed, N., Gamble, R., Bhargava, B., and Linderman, M. 2014. Analysis of End-to-End Cloud Security Protocols with Mobile Devices. In *Smartphone Security and Secure Mobile Cloud Computing (to appear)*, K. Han and J. Kiefer, eds., Springer, NY.

She, W., Yen, I., Thuraisingham, B., and Bertino, E. 2010. Policy-Driven Service Composition with Information Flow Control. In *IEEE International Conference on Web Services*, pp. 50-57.

She, W., Yen, I., Thuraisingham, B., and Huang, S. 2011. Rule-Based Run-Time Information Flow Control in Service Cloud. In *IEEE International Conference on Web Services*, pp. 524-531.

W3C 2004. *Web Services Architecture.* Available from *http* : *//www.w3.org/TR/ws* − *arch/*.

W3C 2008. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* Available from *http : //www.w3.org/TR/xml/.*

NIST 2013. *Special Publication 800-53 Recommended Security Controls for Federal Information Systems Rev. 4.*

Baird, R., and Gamble, R. 2010. Security Controls Applied to Web Service Architectures. In *19th International Conference on Software Engineering and Data Engineering.*

Carlo, D.B., Albers, P., and Hao, J.K. 2006. Web Services Composition. In *Semantic Web Service, Processes and Application*, pp. 195-225.

Hepner, M., Gamble, M.T., and Gamble, R. 2006. Forming a Security Certification Enclave for Service-Oriented Architectures. In *Modeling, Design, and Analysis for Service-oriented Architecture Workshop (MDA4SOA'06).*

Bhargavan, K. et al. 2004. TulaFale: A Security Tool for Web Services. In *International Symposium on Formal Methods for Components and Objects (FMCO).*

Object Management Group 2007. *MOF 2.0 / XMI Mapping Specification, v2.1.1.* Available from *http : //www.omg.org/spec/XMI/2.1.1/.*

Object Management Group 2009. *Service oriented architecture Modeling Language (SoaML): Specification for the UML Profile and Metamodel for Services (UPMS).* Available from *http : //www.omg.org/spec/SoaML/20091101.*

Object Management Group 2011. *UML Profile for CORBA.* Available from *http : //www.omg.org/spec/CORBA/3.1.1.*

Object Management Group 2004. *UML Profile for Enterprise Application Integration (EAI).* Available from *http : //www.omg.org/spec/EAI/1.0/.*

Carlson, D. 2008. *UML Profile for XML Schema.* Available from *http : //www.xmlmodeling.com/documen − tation/specs/XMLSchemaProfile.*

Lautenbacker, F., and Bauer, B. 2007. Creating a Meta-Model for Semantic Web Service Standards. In *19th Proc. of the 3rd Intl. Conf. on Web Information System and Technologies (WEBIST)-Web Interfaces and Applications.*

Thuraisingham, B. 2005. Security Standards for the Semantic Web. In *Computer Standards and Interfaces* Vol. 27, No. 3, pp. 257-268.

Menzel, M., and Meinel, C. 2009. A Security Meta-Model for Service-oriented Architectures. In *IEEE International Conference on Services Computing.*

Dong, F., and Akl, S.G. 2007. An Adaptive Double-layer Workflow Scheduling Approach for Grid Computing. In *21st International Symposium on High Performance Computing Systems and Applications.*

Zheng-qiu, H. et al. 2009. Semantic Security Policy for Web Service. In *IEEE International Symposium on Parallel and Distributed Processing with Applications.*

OASIS 2012. *Web Services Security: SOAP Message Security Version 1.1.1.* Available from *http : //docs.oasis − open.org/wss − m/wss/v1.1.1/os/wss − SOAPMessageSecurity − v1.1.1 − os.html.*

W3C 2001. *Canonical XML.* Available from *http : //www.w3.org/TR/xml − c14n.*

Object Management Group 2010. *Object Constraint Language.* Available from *http : //www.omg.org/spec/OCL/2.3/Beta2/PDF.*

Robert Baird received his B.S., M.S., and Ph.D. in Computer Science at the University of Tulsa. His research areas include Web services, message security, and dyanmic service workflows. He is currently employed by General Dynamics in Pittsburgh, PA.

Rose Gamble is the Tandy Professor in Computer Science and Engineering at the University of Tulsa. She obtained B.S degrees in Mathematics and Computer Science at Westminster College, New Wilmington, PA, and her M.S and D.Sc. degrees in Computer Science at Washington University, St. Louis, MO. Her research areas include Web service and cloud security, cyber trust and suspicion, cyber-physical systems, policy specification and management, software architecture and engineering, game development, and collaborative learning. Her research has been supported by DARPA, OSD, NSF, AFOSR, DoE, AFRL, and Oklahoma state agencies.