MODELLING OF MESSAGE SECURITY CONCERNS WITH UML

Farid Mehr and Ulf Schreier
Faculty of Information Systems, Furtwangen University, Robert-Gerwig-Platz 1, Furtwangen, Germany

Keywords: UML Security, Security Modelling, Security Design, Security Integration, Secure Software Engineering, SOA Security, Model-Driven Security.

Abstract: Service oriented computing is increasingly accepted as a cross-disciplinary paradigm to integrate distributed application functionality through service interfaces. Integration through services as entry points for inter-organisational collaboration can be achieved by exchanging data in messages. In this architectural style, the security of sensitive exchanged data is essential. Security needs to be carefully considered during the entire life-cycle (Devanbu, 2000). Unfortunately, current UML-based modelling approaches do not support the adequate integration of message security concerns. In this paper, we investigate various integration options with UML systematically. The evaluation encompasses most of the options that are proposed today in science and industry as UML profiles. We conclude that neither of those approaches is sufficient for the systematic and comprehensive treatment of message security during modelling. To this end, we propose a new approach that is based on UML and very minor extensions of OCL.

1 INTRODUCTION

This section provides a brief review followed by a description of the contribution of this paper. Afterwards, a sample scenario that is used throughout this paper as well as the organisation of this paper is explained.

1.1 Status Quo

Basic services and composite services, their descriptions as well as operations that utilise or produce those descriptions constitute the foundation of Service-Oriented Architectures (Papazoglou, 2003). Unfortunately, most work on SOA is focusing on the operations at runtime while design methodologies and engineering principles underlying the services have not been considered sufficiently (Papazoglou, 2002).

Due to the abundance of Web Service technologies, Model-Driven Engineering (MDE) approaches are increasingly gaining momentum in the area of service-based applications. One fundamental assumption in MDE is the consideration of models as first-class artefacts (Bézivin, 2005). If security is considered in a MDE approach adequately, the most important security enforcement artefacts as the *glue code*, such as AspectJ code, and the security deployment descriptors, such as a WS-SecurityPolicy document and other files that are specific to the target middleware, can be generated automatically as well. This is important for security because communication links in, for instance, B2B value chains can be based on varying security implementation and/or deployment technologies. Unfortunately, the integration of message security concerns into UML models has not been addressed adequately.

1.2 Contribution

We investigate various approaches for including message security into application models that are expressed in the Unified Modeling Language (UML), version 2.0. UML can be used to model *any* type of a system under study for which it is reasonable to make statements about the *data* maintained and the *behaviour* exhibited by the system (Seidewitz, 2003). It is the de facto standard modelling language, and it is supported by a plenty of modelling tools.

Most of the work that is addressing message security at the model layer is based on the UML profile approach. We conclude that neither of the approaches supported by UML is sufficient for the systematic treatment of message security during modelling, including UML profiles. To this end, we propose a new approach that is based on UML and very minor extensions of the Object Constraint Language (OCL).

This paper focuses on one aspect of security, namely the confidentiality and integrity protection of data that are exchanged between distributed services (Ross, 2001). Hence the term message security because data is carried in messages. Sample messages are SOAP envelopes if a Service-Oriented Architecture is realised with Web Service technologies. Security concerns of data during their processing or of data in storage are not addressed in this paper. Furthermore, other life-cycle phases than modelling, such as implementation, publishing, discovery, selection, enactment, monitoring and adaptation, are not considered in this paper.

1.3 Sample Scenario

We have chosen a well-known order scenario. It is used throughout this paper. An excerpt of the scenario is depicted in Figure 1. Note that the design of the sample scenario in Figure 1 does not yet contain the model of security.

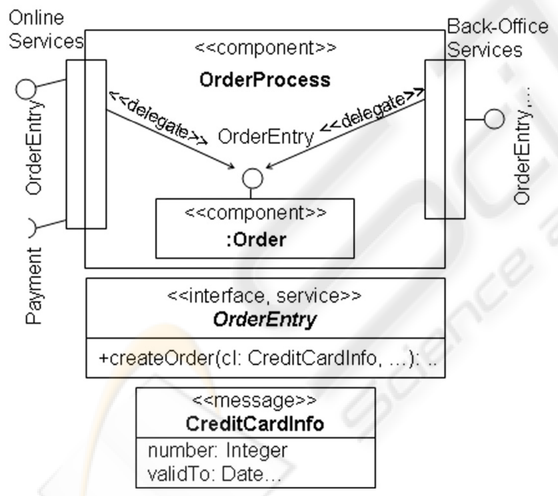


Figure 1: Sample scenario.

A service provider offers the service *OrderEntry*. A sample service provider selling products might be Amazon. *OrderEntry* is offered through the ports *OnlineServices* and *Back-OfficeServices*. It is modelled as an UML interface with the additional stereotype *service*. It provides a single public operation named *createOrder*(..). The exact signature of this operation is not important in this context. Sample input messages are credit card information, order and shipment information. Credit card data are modelled in a class named *CreditCardInfo.* This class is annotated with the stereotype *message*.

1.4 Organisation

The remainder of this paper is organised as follows. Section 2 introduces different message security integration approaches with UML systematically. It begins with the easist approach and ends with the discussion of the most flexible approach that is supported by UML. Section 3 presents the proposed approach. Sections 4 and 5 conclude with discussion on related work.

2 SECURITY INTEGRATION OPTIONS

This section scrutinises different, and mostly complimentary, techniques for integrating message security into application models that are expressed in UML (see also Figure 1).

2.1 UML and OCL as is

In this section, we examine how message security concerns can be modelled with UML and OCL *as is*. Suppose that the integrity and confidentiality are required for credit cards numbers. To this end, the attribute *CreditCardInfo::number* has to be annotated with message security. In this paper, double colons are used to identify composed names. Figure 2 illustrates a sample message security model. It is not complete yet sufficient to delineate the concepts.

Integrity and *Confidentiality* are message security requirements. *MessageSecurityRequirement* is realised by a cryptographic mechanism. Symmetric cryptography may be used to encrypt data. Digital signatures are used to ensure the integrity of data. Asymmetric cryptography is required in order to generate or validate digital signatures.

Cryptography utilises keys. The number of keys that are used depends on the specific cryptraphic mechanism. For instance, symmetric cryptography is based on a shared symmetric key. Additionaly, the number of keys can also depend on the usage context. For instance, during the digital signature creation process, the private key of an entity, such as of a service provider, is required. However, if validating a signature, the private key of the verifying entitiy as well as the public key of the signer is required. Key classes are not depicted in the figure except of the structure of the symmetric key.

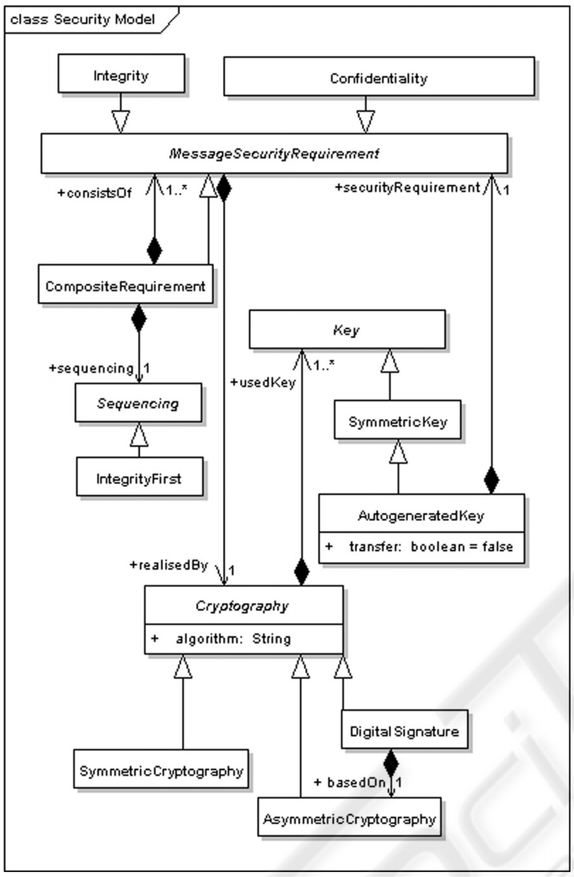


Figure 2: Excerpt of a security model.

An autogenerated key is a symmetric key which is created at runtime. The attribute *AutogeneratedKey::transfer* is used to specify whether a symmetric key has to be transferred to the target service. If true, one can specify security requirements on that key as well. For instance, it might be necessary that an autogenerated key must itself be kept confidential during transmit too.

CompositeRequirement is a composed message security requirement as it consists of more than one requirement. *Sequencing* is used to specify the ordering of computation. For instance, if integrity and confidentiality are required for the same data, *IntegrityFirst* can be used to indicate that the digital signature of that data must be created before its encryption. At the receiver side, the corresponding data are decrypted followed by the verification of the signature of the data.

OCL can be used to restrict the way in which the security model may be utilised. For instance, with the following OCL expression it is formalised that symmetric cryptography is based on one key only (line 4). It has to be of type *SymmetricKey* (lines 5-6). Additionally, it is specified in lines 7-8 that only the algorithms defined in lines 2-3 are allowed.

```
1 context SymmetricCryptography:
2  def: supportedAlgorithms : Set
3 ('Twofish', 'AES',…)
4  inv: self.usedKey->size() = 1 and
5  self.usedKey-> first().
6      oclIsTypeOf(SymmetricKey) and
7  supportedAlgorithms.includes
8            (self.algorithm)
```

Now suppose that confidentiality is required for *CreditCardInfo::number*. The following code excerpt illustrates how OCL can be used to integrate parts of the security model defined previously into *CreditCardInfo::number*.

```
1 context CreditCardInfo inv:
2  self.number.oclIsTypeOf
3       (Confidentiality) and
4  self.number.
5   realisedBy.oclIsTypeOf
6   (SymmetricCryptography)  and
7  self.number.realisedBy.
8      algorithm = 'AES' and
9 self.number.realisedBy.
10     usedKey->first().oclIsTypeOf
11           (AutogeneratedKey) and
12  self.number.realisedBy.
13 usedKey->first().transfer = true
14  and
15  self.number.realisedBy.
16   usedKey->first().
17   securityRequirement.
18      oclIsTypeOf(Confidentiality)
19 and
20   self.number.realisedBy.usedKey
21  ->first().securityRequirement.
22 realisedBy.oclIsTypeOf
23      (AsymmetricCryptography)
24 …
```

It is defined in lines 2-3 that *CreditCardInfo::number* must be of type *Confidentiality*. The specification expressed in OCL constitutes the security *view* on the model element *CreditCardInfo::number*. As depicted in Figure 1, it is also of type *Integer*. In order to model explicitly that *CreditCardInfo::number* is an *Integer* and confidential, one can apply the multiple inheritance technique. For instance, a type named *Both* can be defined. It specialises *Integer* as well as *Confidentiality*. *Both* may now be assigned to

CreditCardInfo::number equally as *Integer* is assigned to it in Figure 1.

CreditCardInfo::number has to be realised by symmetric cryptography (lines 4-6), and the algorithm *AES* has to be applied (lines 7-8). The symmetric key has to be generated at runtime (lines 9-11), and it has to be transferred to the target entity (lines 12-13). The confidentiality of the key has to be ensured as well (lines 15-18). The confidentiality of the key must be realised through asymmetric cryptography (lines 20-23). Equally as illustrated in this example, one can specify the integrity as well as both requirements (see class *CompositeRequirement* in Figure 2).

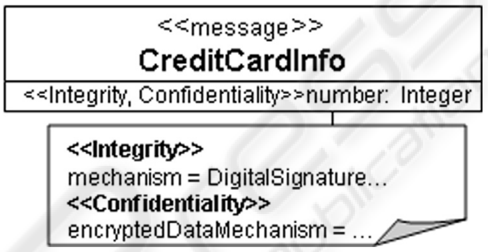
The advantage of this approach is that it is based on standard UML and OCL. Hence, this approach can be applied with any modelling tool that supports UML 2 and OCL. As depicted previously, one can specify composite message security policies as well. For instance, in lines 15-18, confidentiality is specified for the transient symmetric key which in turn is used to ensure the confidentiality of some data.

However, as UML and OCL lack any types of security semantics, the security model in Figure 2 as well as the OCL expression cannot be validated semantically. This is true because the security model in Figure 2 constitutes a security domain model that is not expressed in a security Domain Specific Language (DSL). It is expressed in UML instead. Futhermore, the model in Figure 2 together with the OCL statements cannot be used in transformation rules to generate the security artefacts out of the integrated models in a MDE process. This is true because transformation rules should be written against metamodels and applied to models. The security model and the OCL code are expressed in languages that suffer security constructs. Consequently, this approach can only be used to document software.

2.2 UML Profiles

UML profile is a standard extension mechanism. It facilitates the definition of new dialects. Any metaclasses from the UML metamodel can be extended toward a particular domain, platform or method. This is not a first-class extension mechanism because it does not allow a metamodel to be modified. The referenced metamodel is present in *read-only* mode instead.

Profiles are specific types of packages carrying limited kinds of metaclasses (called stereotypes) and metaatributes (named tag definitions or stereotype properties). Figure 3 depicts the application of an excerpt of a message security profile. The requirements integrity and confidentiality are attached to *CreditCardInfo::number* through corresponding stereotypes. In this example, values are assigned to stereotype properties in a comment block. The properties provide detailed instructions on how to implement the requirements. That is, stereotypes are used to indicate requirements (*what*), whereas properties are used to specify *how* to enforce the requirements, but with the operational data only that are relevant at the modelling stage.



<<message>>
CreditCardInfo
<<Integrity, Confidentiality>>number: Integer
<<Integrity>>
mechanism = DigitalSignature...
<<Confidentiality>>
encryptedDataMechanism = ...

Figure 3: Application of a message security profile.

The main advantage of this approach is that stereotypes and their properties can drive the transformation rules. In the industry, there is a broad tool support for the profile mechanism too. Many authors have been applying this approach. For instance, stereotypes that are specific to the Web Service technology, such as *WebServiceCall*, and properties, as *portType* or *wsdl,* are proposed in (Skogan, 2004). The generation of WSDL and XML schema documents out of stereotyped UML classes is discussed in (Grønmo, 2004). A few basic security stereotypes are defined in (UML Working Group, 2006b). For Service-Oriented Architectures, an UML profile is defined in (Wada, 2006). In this single profile, fundamental concepts of a SOA, such as services and messages, are defined together with message security concerns as encryption algorithm. This profile is applied during the modelling phase exactly as described in this section.

However, this meta-modelling technique is very limited. For instance, it is very hard to model composition relations between stereotypes. The composition is especially important for security because security concerns can be related to non-functional properties, including security, as well. For instance, how can the modeller express the following two security intentions with the UML profile approach: “the password that is used for authentication should be encrypted according to a specific policy”.

Additionally, the separation of concerns principle is weakened heavily with this approach because security specifications are *scattered* across application models. For instance, in Figure 3, *CreditCardInfo::number* is *tangled* with integrity and confidentiality concerns. It is not possible to specify message security policies separately. Therefore, a single policy cannot be applied to (i.e., reused in) several model elements. As a consequence, in large application models, the management of the security policies becomes a tough problem. For instance, the change of a security policy can require its alteration at every place where it is applied through stereotypes and stereotype properties. Additionally, the modified model(s) must be regenerated.

The next section investigates a complimentary approach. It bypasses the problem of *tangling* as discussed previously.

2.3 UML Templates

UML's package *Templates* provides *advanced* (de)composition capabilities for models. This section investigates the suitability of UML templates for the integration of message security.

UML templates support the *isolated* modelling of *any* types of concerns such as non-functional or technological concerns. Templates are composed into the architecture of an application through *binding*. Binding is a task which replaces the specified parameters of a template with the actual parameters of an application model. Templates have in UML a similar function as aspects in Aspect-Oriented Programming (Clarke, 2001).

Figure 4 illustrates the binding of message properties. Area *A* contains the models that form the functional part of an application. Security is specified in area *B* similarly as presented in the previous sections. However, the model elements in area *B* are independent of the model elements in area *A*. Consequently, the security specifications in *B* are reusable between different application models. To recapitulate, this is not possible with the UML profile mechanism which is widely applied today. The composition into a particular application model (i.e., reuse) is carried out in area *C*.

In this example, *MessageTemplate* represents a message template. *MessageTemplate::attribute1* and *MessageTemplate::attribute2* constitute formal template parameters. These are annotated with security stereotypes and stereotype properties in the same manner as introduced in the previous section. That is, this approach is complementary to UML profiles. Area *B* indicates *what* to do, such as "enforce integrity", and *how* to achieve it, for instance "use digital signatures". In area *C*, it is specified *where* to apply the security model(s). In this instance, *CreditCardInfo::number* is bound to both parameters of the message template.

The sequencing can be specified either explicitly, for instance as an additional stereotype, or implicitly. For instance, the binding sequence of the formal parameters in area *C* can imply the sequencing. In this case, integrity must be enforced first followed by confidentiality.

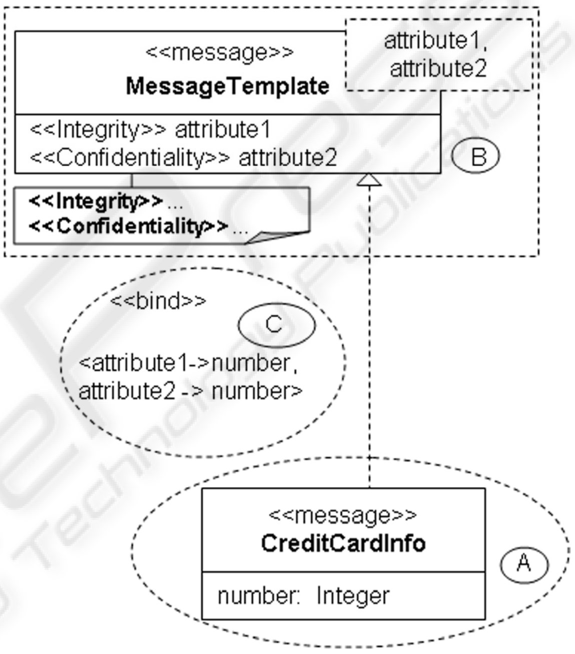


Figure 4: (De)Composition of structural models.

An advanced application of UML templates for message security is depicted in Figure 5. It considers additionally behavioural concerns during the specification of security in area *B* such as the flow of data and execution. The description in this figure is incomplete but sufficient to impart concepts. Syntactic details are omitted as well.

SecureService represents a service template. Its single operation is exposed as a formal template parameter. The activity *BusinessTask* specifies the *template* flow for the operation. It consists of three actions. Each action constitutes a call to a behaviour specification. The second action calls a behaviour named *MainProcess* which is exposed as a formal parameter as well.

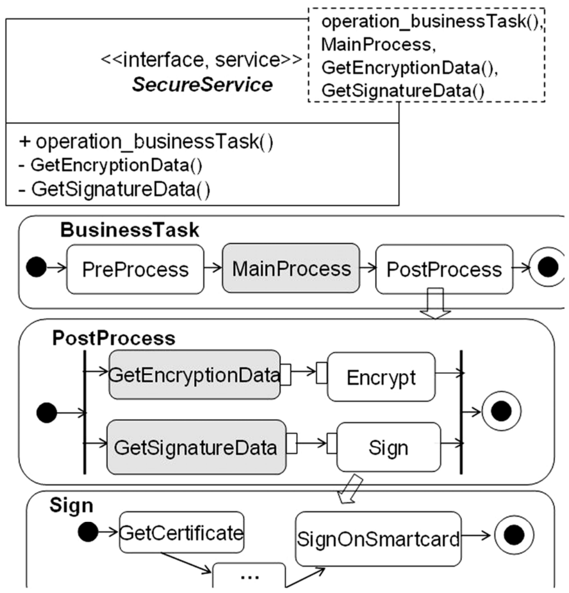


Figure 5: A service template.

The lower area in Figure 5 represents the two activities that are connected with the actions of the same name. *PostProcess* consists of two concurrent flows. The upper (lower) flow encrypts (signs) data. The corresponding sensitive data are obtained through the actions *GetEncryptionData* and *GetSignatureData*. These represent calls to operations of the same name. Those operations are declared as template parameters as well.

The lowest area in Figure 5 depicts parts of the sample activity *Sign*. It contains the sequencing of actions and the flow of data for a signature creation process. The detailed security properties for each action can be specified in a flat list through stereotype properties similarly as described in section 2.2.

Figure 6 shows the binding of the template *SecureService* with the actual data of the service *OrderEntry*. Assume that *OrderInfo* contains some order information as the ordered items and shipping address.

OrderEntry::createOrder(..) is bound to the single operation of the service template. The behaviour *OrderEntry::ProcessOrder* is bound to *SecureService::MainProcess*. Assume that *ProcessOrder* constitutes a behaviour specification as an activity. It is owned by *OrderEntry*. The other two private operations defined in *OrderEntry* are used to provide the sensitive data that are required in the corresponding two activity actions in area *B*.

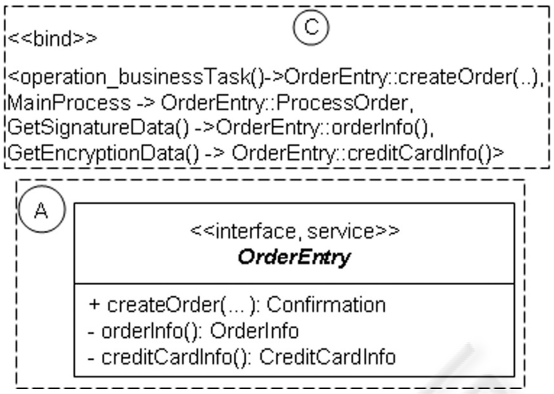

C
<<bind>>
<operation_businessTask()->OrderEntry::createOrder(..),
MainProcess -> OrderEntry::ProcessOrder,
GetSignatureData() ->OrderEntry::orderInfo(),
GetEncryptionData() -> OrderEntry::creditCardInfo()>
A
<<interface, service>>
OrderEntry
+ createOrder(...): Confirmation
- orderInfo(): OrderInfo
- creditCardInfo(): CreditCardInfo

Figure 6: Binding of OrderEntry.

The first approach presented in Figure 4 supports the isolated modelling of (cross-cutting) concerns. Message security properties are specified in structural models in area *B*. This approach is complementarily in so far as message security can be modelled in *B* according to every option discussed earlier, if needed.

Area *B* is independent of any application models because this area is concerned primarily with the questions *what* to do and *how* to enforce security, but not *where* and *when* to enforce security. The composition is established in area *C*. It indicates *where* to apply security. This is achieved through parameterisation. In addition, a further dimension can be defined that governs *when* a specific composition of the models is valid. This subject is not addressed for UML templates in this paper.

The second approach presented in Figure 5 is more flexible because behavioural elements are defined as formal template parameters additionally.

The UML templates mechanism facilitates reuse and evolution of security specifications in area *B* because those are maintained in their own space. However, this approach can lead to more complex models, especially in area *B*, as can be inferred from Figure 5. Consequently, without a systematic approach that is also supported by tools, the comprehensibility and manageability of the application architecture can suffer significantly.

Another disadvantage is that the models in area *A* cannot always be isolated completely from the data in area *B*. For instance, as illustrated before, the operations *OrderEntry::orderInfo()* and *OrderEntry::creditCardInfo()* must be designed in area *A* explicitly because these are *required* in area *B*. That is, area *A* has to be designed with security in mind (i.e., design *for* security). Consequently, this approach is not sufficient for the systematic treatment of message security either.

3 PROPOSED APPROACH

This section begins with a sample policy followed by a description of the proposed integration approach. Afterwards, different integration scenarios based on this approach are presented.

3.1 A Message Security Policy

This section presents a sample and simple policy in order to show our integration approach in the subsequent sections. The policy is presented in UML 2 notation. It is not the purpose of this paper to introduce a full-fledged policy language as the main focus here is on the subject of integration.

Figure 7 shows the black-box view of a message security policy for the domain *Payment* (see also Figure 1). The policy resides in area *B* (according to the template-based approach). In the next two figures, colons are placed immediately before the type of instance specifications. The optional names of instance specifications are placed before colons. Equal signs are used to assign values to attributes. Values that can be defined unrestrictedly are placed in single quotation marks. Commas are used to separate values. Semicolons are used to detach attributes if these are carried in the same line.

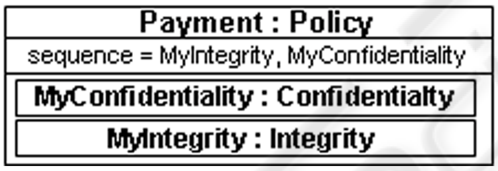

Payment : Policy
sequence = MyIntegrity, MyConfidentiality
MyConfidentiality : Confidentialty
MyIntegrity : Integrity

Figure 7: Policy Payment.

The policy begins with an identifier (*Payment*). *Payment* consists of the two specifications *Payment::MyIntegrity* and *Payment::MyConfidentiality*. These can be applied to application models together.

If *Payment* is referenced from a model element as a whole, its integrity has to be enforced followed by its confidentiality. This information is carried in the attribute *sequence* which is displayed in the first compartment of *Payment*.

For illustrative purposes, the specification *Payment::MyConfidentiality* is depicted in Figure 8. The specification of *Payment::MyIntegrity* is not presented in this paper.

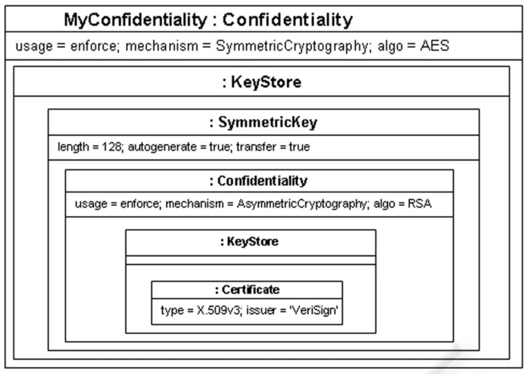

MyConfidentiality : Confidentiality
usage = enforce; mechanism = SymmetricCryptography; algo = AES
: KeyStore
: SymmetricKey
length = 128; autogenerate = true; transfer = true
: Confidentiality
usage = enforce; mechanism = AsymmetricCryptography; algo = RSA
: KeyStore
: Certificate
type = X.509v3; issuer = 'VeriSign'

Figure 8: Payment::MyConfidentiality.

The three attributes indicate that confidentiality has to be enforced through symmetric cryptography and the algorithm *AES*.

All instance specifications in *Payment::MyConfidentiality* are unnamed because these are not supposed to be referenced from elsewhere. Symmetric key data are defined next. A symmetric key has to be generated at runtime automatically. It must have a length of 128 bits. The key has to be transferred to the target entity of the corresponding message.

As the key must be kept confidential during transmit, it contains an *inner*, *anonym* confidentiality specification. The nesting mechanism enables to qualify specific aspects of another assertion. Here, the symmetric key has to be encrypted by means of the asymmetric algorithm *RSA*. The corresponding certificate data are defined too.

3.2 Integration Approach

Security policies are specified separately as in the UML template approach. However, compared to UML templates, security policies do not need to be *modelled*. One can leverage a security language for that purpose instead. For instance, sample textual policies expressed in XML and WS-SecurityPolicy can be found in (Nakamura, 2005) and (Tatsubori, 2004). Not all kinds of information related to security should be specified in the design phase. For instance, operational level data are relevant as the security enforcement mechanism. However, deployment level information should be carried in the deployment phase such as the filename of a key store. In section 3.1, we have presented a sample policy in the UML 2 notation.

The integration of message security policies into application models is based on OCL with very minor extensions. The ability to own constraints, including constraints expressed in OCL, has been considered thoroughly during the design of the UML metamodel. For instance, the metaclass *Namespace* is capable of owning constraints. The constraints can apply to elements in the *Namespace*. For instance, as *Class* is a *Namespace*, constraints owned by *Class* can apply to its named elements like properties and operations. Next sections discuss the integration into some kind of model elements that can be used to represent concepts in a Service-Oriented Architecture. Due to space limitations, only a few examples are given to illustrate the approach. We will begin with messages.

3.3 Integration into Message Classes

Message security policies can be attached to messages directly. In this option, the integrated policies have a global scope; that is, policies are enforced at *each* remote interaction point. This option should be chosen if the message security policies for the sensitive information carried in a message class is identical for each interaction point. This might be the case if the system under study collaborates with other entities through insecure protocols only as SOAP and HTTP.

CreditCardInfo contains sensitive attributes. Suppose that integrity and confidentiality are required for *CreditCardInfo::number.* Moreover, integrity is demanded for *CreditCardInfo::validTo.* Integrity and confidentiality are specified in the policy *Payment* (see Figure 7 and Figure 8). The policy and parts thereof are integrated into messages with OCL as follows.

```
context CreditCardInfo inv:
self.number.isSecure("Payment")
   and
self.validTo.
   isInteger("Payment.MyIntegrity")
```

This constitutes the area *C* of the UML template approach. The keyword *context* introduces the context of this expression which is the message class *CreditCardInfo*. The keyword *inv* indicates that the following constraints, which are of type Boolean, must be true for all instances of *CreditCardInfo*. The reserved word *self* refers to *the* contextual instance.

CreditCardInfo::number is declared to be "secure" according to the policy *Payment*. It is specified in the policy that integrity has to be enforced followed by confidentiality (see Figure 7).

CreditCardInfo::validTo has to be integer according to the sub-specification *Payment::MyIntegrity*.

The type of enforcement in an interaction point depends on the current communication path. For instance, *CreditCardInfo::number* will be signed and encrypted on the way from client to server (output channel). Conversely, when arriving in a response message (input channel), it has to be decrypted and the signature needs to be validated. To recapitulate, in this integration option the enforcement will be performed globally, namely in the context of the overall application, independent of the services that are exchanging credit card information.

3.4 Integration into Services

Security policies can be attached in the context of a service operation. This option should be used if the message security policies for a message vary in different operations. This is exemplified next.

Suppose that *CreditCardInfo* is utilised in *OrderEntry::createOrder(oI : OrderInfo)* indirectly, through the message class *OrderInfo*. Indirectly means that a type is not specified as an input parameter of an operation directly but is navigable from one of the directly declared input parameters. In this example, *OrderInfo* contains a property (association) named *creditCardInfo* of type *CreditCardInfo*.

Additionally, assume that *CreditCardInfo* is required in service *Payment* indirectly, through *Payment::book(..)*. Furthermore, assume that *Payment* is used over secure networks whereas *OrderEntry* is provided over insecure networks (see ports in Figure 1). In this case, message security for *CreditCardInfo* in the context of *OrderEntry* must be enforced differently than for *Payment.*

The following OCL expression integrates message security policies into the same message class as in the previous section but in the context of the service operation *OrderEntry::createOrder(oI : OrderInfo)* only.

```
   context OrderEntry::createOrder(oI :
OrderInfo)
   pre:
   oI.creditCardInfo.number.isSecure("P
ayment")
      and
   oI.creditCardInfo.validTo.
      isInteger("Payment.MyIntegrity")
```

The keyword *context* refers to the classifier possessing this operation. As the operation is included in *OrderEntry*, this service constitutes the classifier. The keyword *pre* indicates that the following two conditions represent pre-conditions. Message security is integrated into *OrderInfo::creditCardInfo* identically as in section 3.3. Hence, it is specified that before the operation executes, *CreditCardInfo::number* must be decrypted, and its digital signature has to be validated. The integrity of *CreditCardInfo::validTo* has to be validated likewise. Post-conditions can be specified similarly, by putting the standard label *post* before the actual post-conditions.

3.5 Integration into Ports

If a policy is attached in the context of a service, which is the case in the previous two options, the security policy is enforced for each *entity* implementing that service. However, depending on the current port, the message security policies can vary.

As depicted in Figure 1, the service *OrderEntry* is provided at two ports. Suppose that *OnlineServices* corresponds to the SOAP *interface*. All provided (required) services at this port are offered (demanded) as Web Services. Likewise, assume that port *Back-OfficeServices* corresponds to the protocol RMI/IIOP. Message security for *Back-OfficeServices* is not required if this port is used internally and/or in a secure environment. In this case, message security policies should be attached in the context of ports. The next sample code snippet illustrates how to specify security for *CreditCardInfo* as in the previous section but for the port *OnlineServices* only.

```
context OrderProcess inv:
self.OnlineServices.OrderEntry.cI.nu
mber.isSecure("Payment")
    and
self.OnlineServices.OrderEntry.cI.va
lidTo.isInteger("Payment.MyIntegrity")
```

The component *OrderProcess* represents the context classifier; that is, it owns the expression (see also Figure 1). The keyword *self* is utilised as before; to reference named elements that are owned by the classifier. The whole policy *Payment* and the sub-specification *Payment::MyIntegrity* are applied to the two attributes of *CreditCardInfo* as before.

Ports are not restricted to components, but can be employed with any model element that is of type *EncapsulatedClassifer*. For instance, it is possible to model classes that provide or require (the same) services at different interaction points as well. Hence, our approach is not restricted to the modelling of components providing and/or requiring services.

4 RELATED WORK

An UML extension for security called UMLSec is proposed in (Jürjens, 2002). In this approach, model elements are annotated with stereotypes and stereotype properties for capturing security policies, as introduced in section 2.2. A model centric approach for specifying and integrating access control policies is presented in (Lodderstedt, 2002). From the architecture point of view, a connector-centric approach is introduced in (Ren, 2005). We investigate various mechanisms for including message security concerns into application models in a systematic way, including UML profiles.

Initial research work toward the modelling of service-based applications can be found in (Manolescu, 2005), (Skogan, 2004) and (Baina, 2004). However, these authors disregard any kinds of security concerns. Consequently, security is considered as an afterthought mostly. An UML profile including SOA concepts through stereotypes, such as message and service, and security aspects through stereotype properties, as encryption algorithm, is proposed in (Wada, 2006). We evaluate the UML profile mechanism in section 2.2 and conclude that it is not sufficient for the comprehensive treatment of message security. To this end, we propose an advanced approach which is based on OCL. It is complementary to UML profiles and UML templates.

Recent research results in the area of separation of concerns on the level of modelling can be obtained in, among others, (Gray, 2003), (Clarke, 2001), (Katara, 2003) and (Jacobson, 2004). Neither of them investigates the suitability to message security as has been done in this paper. Message security on the level of program code is considered in (Baligand, 2004).

5 CONCLUSION

The security of sensitive exchanged data has to be addressed in Service-Oriented Architectures. Message security needs to be integrated into earlier engineering phases. Unfortunately, current approaches do not support this adequately as most work is focused on UML profiles. In this paper we investigated several approaches that are applicable with the UML systematically, by beginning with the easiest approach and ending with UML templates combined with UML profiles. Based on the findings during the evaluation, we conclude that neither of the approaches provided by UML is sufficient for the systematic treatment of message security during modelling.

To this end, we propose a new approach that leverages the capability of UML model elements to own constraints expressed in OCL. As the proposed approach does not entail any proprietary extensions to UML it can be implemented in any modelling tools that support UML 2. The proposed approach can be applied complementarily to UML profiles and UML templates, if necessary.

REFERENCES

Baligand, F., Monfort, V., 2004. A concrete solution for web services adaptability using policies and aspects. In *Proceedings of the 2nd International Conference on Service Oriented Computing*. ACM Press, NY, USA, pp. 134-142.

Baina, K., Benatallah, B., Casati, F., Toumani, F., 2004. Model-Driven Web Service Development. In *Proceedings of the 16th International Conference on Advanced Information Systems Engineering*. Springer, Berlin/Heidelberg, Germany, pp. 290-306.

Bézivin, J., Devedžić, V., Djurić, D., Favreau, J., Gašević, D., Jouault, F., 2005. An M3-Neutral infrastructure for bridging model engineering and ontology engineering. In *Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications*. Springer, Germany, pp. 159-171.

Clarke, S., Walker, R.J., 2001. Composition patterns: an approach to designing reusable aspects. In *Proceedings of the 23rd International Conference on Software Engineering*. IEEE Computer Society, Washington, DC, USA, pp. 5-14.

Devanbu, W.T., Stubblebine, S., 2000. Software Engineering for Security: a Roadmap. In Proceedings of *Conference on the Future of Software Engineering*. ACM Press, New York, USA, pp. 227-239.

Gray, J., Bapty, T., Neema, S., Schmidt, D.C., Gokhale, A., Natarajan, B., 2003. An approach for supporting aspect-oriented domain modelling. In *Proceedings of the second international conference on Generative programming and component engineering*. Springer, NY, USA, 2003, pp. 151-168.

Grønmo, J., Skogan, D., Solheim, I., Oldevik, J., 2004. Model-driven Web Services Development. In *IEEE International Conference on e-Technology, e-Commerce and e-Services*, eee, 2004, pp. 42-45.

Jacobson, I., Ng, P., 2004. *Aspect-Oriented Software Development with Use Cases*. Addison Wesley, NY, USA.

Jürjens, J., 2002. UMLSec: Extending UML for Secure Software Development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*. Springer, London, UK, pp. 412-425.

Katara, K., Katz, S., 2003. Architectural Views of Aspects. In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*. ACM Press, NY, USA, 2003, pp. 1-10.

Lodderstedt, T, Basin, D.A., Doser, J., 2002. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of the 5th International Conference on The Unified Modeling Langauge*. Springer, London, UK, pp. 426-441

Manolescu, I., Brambilla, M., Ceri, S., Comai, S., Fraternali, P., 2005. Model-driven design and deployment of service-enabled web applications. In *ACM Transactions on Internet Technology*. ACM Press, NY, USA, volume 5, issue 3, pp. 439-479.

Nakamura, Y., Tatsubori, M., Imamura, T., Ono, K., 2005. Model-Driven Security Based on a Web Services Security Architecture. In *Proceedings of the IEEE International Conference on Services Computing IEEE Computer Society*, NY, USA, volume 1, pp. 7-15.

Papazoglou, M.P., Yang, J., 2002. Design Methodology for Web Services and Business Processes. In *Proceedings of the Third International Workshop on Technologies for E-Services*. Springer, London, UK, pp. 54-64.

Papazoglou, M.P., Georgakopoulos, D., 2003. Service-Oriented Computing. In *Communications of the ACM*. ACM Press, New York, USA, volume 46, issue 10, pp. 24-28.

Ren, J., Taylor, R., Dourish, P., Redmiles, D., 2005. Towards an Architectural Treatment of Software Security: a Connector-Centric Approach. In *Proceedings of the 2005 workshop on Software engineering for secure systems-building trustworthy systems*. ACM Press, New York, USA, pp. 1-7.

Ross, A., 2001. *Security Engineering. A Guide to Building Dependable Distributed Systems*, John Wiley & Sons, New York, Chichester, Weinheim.

Seidewitz, E., 2003. What Models Mean. In *IEEE Software*. IEEE Computer Society Press, Los Alamitos, CA, USA, volume 20, issue 5, pp. 26-32.

Skogan, D., Gronmo, R., Solheim, I., 2004. Web Service Composition in UML. In *Proceedings of the Enterprise Distributed Object Computing Conference.* Eight IEEE International, IEEE Computer Society, Washington, DC, CA, USA, pp. 47-57.

Tatsubori, M., Imamura, T., Nakamura, Y., 2004. Best-Practice Patterns and Tool Support for Configuring Secure Web Services Messaging. In *Proceedings of the International Conference on Web Services*. IEEE Computer Society, Washington, DC, CA, USA, pp. 244-251.

UML Working Group, 2006b, UML Profile For Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. Version 1.0, OMG document number: formal/06-05-02.

Wada, H., Suzuki, J, Oba, K., 2006: A Service-Oriented Design Framework for Secure Network Applications. In *Proceedings of the 30th IEEE International Conference on Computer Software and Applications Conference*. Chicago, IL, USA, volume 00, pp. 359-368.