A Service-Oriented Design Framework for Secure Network Applications

Hiroshi Wada and Junichi Suzuki
Department of Computer Science
University of Massachusetts, Boston
Boston, MA 02125-3393
{shu, jxs}@cs.umb.edu

Katsuya Oba
OGIS International, Inc.
Palo Alto, CA 94301
oba@ogis-international.com

Abstract—*Service Oriented Architecture (SOA) is an architectural style to reuse and integrate existing systems for designing new applications. Each application is designed in an implementation independent manner using two major abstract concepts: services and connections between services. In SOA, the non-functional aspects (e.g., security and fault tolerance) of services and connections should be described separately from their functional aspects (i.e., business logic) because different applications use services and connections in different non-functional contexts. This paper presents a UML profile to graphically describe and maintain non-functional aspects in SOA in an implementation independent manner. This paper also describes how the proposed profile is used to develop secure service-oriented applications.*

1. Introduction

One of the current key issues in large-scale distributed systems is to reuse and integrate existing systems to build new applications in a cost effective manner [1, 2]. Service Oriented Architecture (SOA) addresses this issue by improving the reusability and maintainability of distributed systems [3, 4]. It is an architectural style to design applications in an implementation independent manner using two major abstract concepts: *services* and *connections between services*. Each service encapsulates the function of a subsystem in an existing system. Each connection defines how services are connected with each other and how messages are exchanged through the connection. SOA hides the implementation details of services and connections (e.g., programming language and remoting middleware) from application designers. They can reuse and combine services with connections to build their applications without knowing the implementation details of services and connections.

In SOA, the non-functional aspects (e.g., security and fault tolerance) of services and connections should be defined separately from their functional aspects (i.e., business logic) because different applications use services and connections in different non-functional contexts. For example, an application may send signed and encrypted messages to a service when the messages travel to a destination service through third-party intermediaries, in order to prevent the intermediaries from maliciously sniffing or altering the messages. Another application may send plain messages to the service via unsecured connection when the service is hosted in-house. The separation of functional and non-functional aspects improves the reusability of services and connections. It also allows two different aspects to evolve independently, and improves the ease of understanding application design. This contributes to improve the maintainability of applications.

This paper proposes a Unified Modeling Language (UML) profile to graphically model the non-functional aspects in SOA, including security aspects, as UML diagrams (composite structure diagrams and class diagrams). A UML profile specializes the standard UML model elements (e.g., class and association) to precisely describe domain specific or application specific concepts [5, 6]. Using the proposed UML profile, non-functional aspects in SOA can be modeled without depending on any particular implementation technologies. Supporting tools accept the UML models defined with the proposed profile and transform them into application code using certain implementation technologies such as Enterprise Service Buses (ESBs) [7] and secure file transfer systems. This paper focuses on the design details of the proposed UML profile, and describes how it is used to develop secure service-oriented applications.

This paper is structured as follows. Section 2 motivates the proposed UML profile with an example, and Section 3 describes the design details of the proposed profile. Section 4 demonstrates how the proposed profile is used in a service-oriented application development. Sections 5 and 6 conclude with discussion on related work and future work.

2. Background and a Motivating Example

UML is a modeling language to describe application designs as graphical diagrams. It specifies the syntax (or notation) and semantics of every model element that appears in diagrams (e.g., class, interface and association). The syntax and semantics are defined in the UML metamodel [5],

which is the grammar specification for a standard (default) set of model elements in UML.

In addition to standard model elements, UML provides extension mechanisms (e.g., stereotypes and tagged-values) to specialize the standard model elements to precisely describe domain or application specific concepts [6]. A stereotype is applied to a standard model element, and specializes its semantics to a particular domain or application. Each stereotyped model element can have data fields, called tagged-values, specific to the stereotype. Each tagged-value consists of a name and value. A particular set of stereotypes and tagged-values is called a UML profile.

Figure 1 shows an example model defined with the proposed UML profile. It illustrates a loan application processing in which a bank assesses whether to grant a loan to an applicant based on the applicant's credit rating evaluated by a credit agency. In this example, three services (`Applicant`, `Bank` and `CreditAgency`) exchange messages. Each service is defined as a class decorated with the stereotype ≪`Service`≫. Services exchange four types of messages (`LoanApp`, `LoanApproval`, `RatingInquiry` and `Rating`), each of which is stereotyped with ≪`Message`≫. The data fields steteotyped with ≪`EncryptedProperty`≫ in `RatingInquiry` and `Rating` are encrypted with the algorithms specified as tagged-values (`algorithm = ...`).

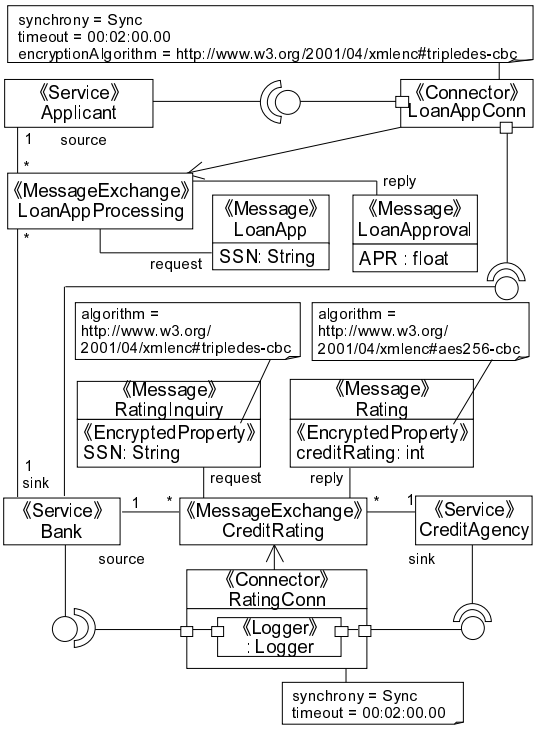

Figure 1. An Example UML Model

Each pair of a request and reply messages is represented by ≪`MessageExchange`≫. ≪`Connector`≫ represents a connection that transmits messages between services. In this example, messages are delivered through two connectors (`LoanAppConn` and `RatingConn`). Every message exchange is bound with a connector in order to specify which connector is used to deliver messages. Figure 1 shows that an `Applicant` sends a `LoanApp` message through `LoanAppConn` to a `Bank`, which in turn sends a `RatingInquiry` message through `RatingConn` to a `CreditAgency`. A `CreditAgency` sends a `Rating` back to a `Bank`, which returns a `LoanApproval` to an `Applicant`. Each connector can have multiple tagged-values to specify message transmission semantics. In this example, `LoanAppConn` specifies the synchrony of message transmission (synchronous) and the timeout of message transmission (two minutes).

As shown above, the proposed UML profile provides a visual and intuitive abstraction to model the architectures and non-functional aspects of service-oriented applications.

3. Design of the Proposed UML Profile

The proposed UML profile provides key model elements to specify service-oriented applications: *service*, *message exchange*, *message*, *connector* and *filter*, each of which is defined as stereotypes (Table 1).

Table 1. Stereotypes

Stereotype	Description
Service	Represents a service.
MessageExchange	Represents a pair of a request and reply messages. Specifies which services send and receive the messages.
Message	Represents a (request or reply) message.
Connector	Represents a connection between services (i.e., message source and destination). Defines the semantics of message transmission and processing. Specifies which messages (message exchange) to transmit.
Filter	Specifies the semantics of message transmission and message processing in each connector.

Figure 2 shows how the proposed UML profile defines its stereotypes by extending the UML metamodel. Each stereotype is defined as a metaclass stereotyped with ≪`stereotype`≫. Except `Connector`, four stereotypes inherit the `Class` metaclass in the `Kernel` package of the UML metamodel. Thus, they are applied to classes in user-defined models (see Figure 1). A `Service` can be a source or sink of each request/reply message. The source and sink are identified with `source` and `sink`, roles on two associations between a `MessageExchange` and `Services` (Figures 1). Each `MessageExchange` may have multiple re-

ply messages per request message (Figure 2). Using multiplicity on two associations between a `MessageExchange` and `Services`, `MessageExchange` can indicate one-to-one (unicast) and one-to-many (multicast or manycast) message exchanges. For example, Figure 1 shows a one-to-one message exchange between an `Applicant` and a `Bank`.

`Connector` is a stereotype extending the `Class` metaclass in the `InternalStructures` package of the UML metamodel (Figure 2). This metaclass defines a composite class, a special type of class, which can contain other model elements (e.g., inner classes)[1] and have `Ports` to specify how internal model elements interact with external elements. In the proposed UML profile, a `Connector` can contain `Filters` to specify the semantics of message transmission and message processing. The `Ports` connected with a `Connector` identify the `Messages` it receives and sends out, using association roles `input` and `output`. For example, Figure 1 shows the connector `RatingConn`, which contains a filter (a `Logger`). This filter receives, records and sends out `creditRating` messages.

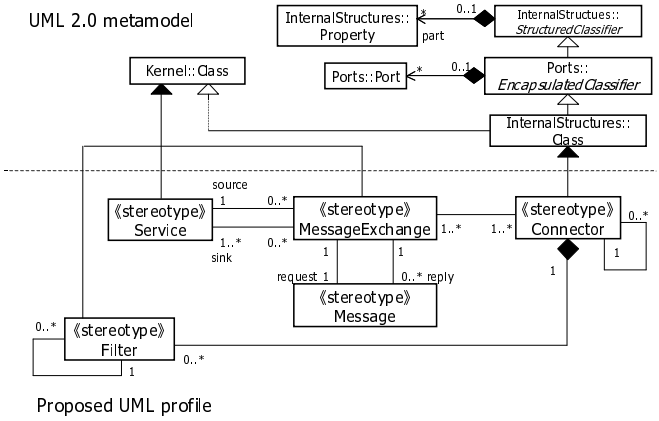

Figure 2. Definition of Stereotypes

3.1. Connector

`Connector` has five tagged-values (Figure 3). `timeout` is a mandatory tagged-value to specify the timeout period (in millisecond) in which a connector needs to deliver each message. If a message is not delivered to its destination (sink) within the timeout period, a connector discards the message. In Figure 1, the timeout period of the connector `LoanAppConn` is specified as two minutes.

`synchrony` is a mandatory tagged-value to specify the synchrony semantics of message transmissions between a message source and destination. Synchronous, asynchronous and oneway non-blocking semantics are defined as an enumeration in `Synchrony` (Figure 3), and each connector chooses one of them. In Figure 1, `Applicants` and

[1]Precisely, a composite class can contain any *classifiers*, defined in the UML metamodel.

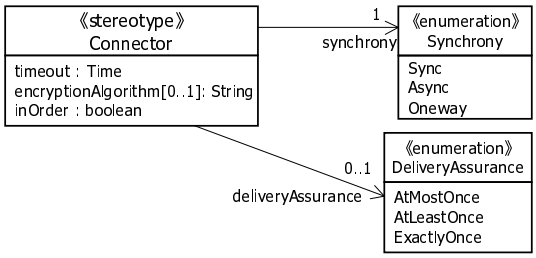

Figure 3. Tagged-Values of Connector

`Banks` exchange `LoanApp` and `LoanApproval` messages synchronously.

`inOrder` is a mandatory tagged-value to specify whether the order of messages that a service (message destination) receives is same as the order of messages that the other service (message source) sends out. The default value of `inOrder` is false.

`deliveryAssurance` is an optional tagged-value to specify the assurance level of message delivery. Three different semantics are defined as an enumeration in `DeliveryAssurance` (Figure 3), and each `Connector` chooses one of them at a time (see Figure 1). `AtLeastOnce` means that a connector retries delivering a message until its destination receives the message. (A message retransmission is triggered with the `timeout` tagged-value.) However, the message may be delivered to its destination more than once. `AtMostOnce` means that a connector discards a message if the message has already been delivered to its destination; however, there is no guarantee of message delivery. `ExactlyOnce` satisfies the requirements of the above both semantics. It guarantees that a connector delivers a message to its destination without duplications. When `inOrder` is true, `ExactlyOnce` is implicitly (automatically) set to `deliveryAssurance` because duplicated or missing messages violate the `inOrder` semantics.

Figure 4 shows an example model using `inOrder` and `deliveryAssurance`. This example illustrates an extension to an order processing application in Figure 1. In this example, a `Buyer` transmits an `OrderMsg` to a `Supplier` via `Retailer`. After a `Retailer` forwards an `OrderMsg` from a `Buyer` to a `Supplier`, the `Buyer` can cancel the order by transmitting a `CancellationMsg` to the `Retailer`, and in turn, to the `Supplier`. In this example, the order of message transmissions is important between `Retailer` and `Supplier` because an order must be delivered to `Supplier` before a corresponding order cancellation. Therefore, the `inOrder` semantics is assigned to the `RetailerOrder` connector. This semantics implicitly assigns `ExactlyOnce` to the `deliveryAssuarance` semantics in the `RetailerOrder` connector.

`encryptionAlgorithm` is an optional tagged-value used for transport-level (point-to-point) encryption in a

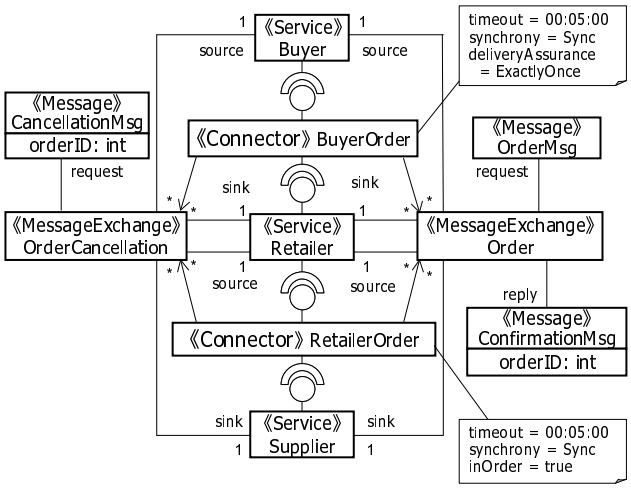

Figure 4. An Example of inOrder and deliveryAssurance

connector. (see Section 3.4 for message-level (end-to-end) encryption) This tagged-value defines an algorithm to secure a connection upon which request and reply messages are transmitted. The encryption algorithm is specified as a URI defined in the XML Encryption specification [8]. For example Triple DES is represented with http://www.w3.org/2001/04/xmlenc#tripledes-cbc, and AES-256 (Advanced Encryption Standard) is represented with http://www.w3.org/2001/04/xmlenc#aes256-cbc. In Figure 1, the connector `LoanAppConn` uses Triple DES.

3.2. Filter

This paper describes five of the filters that the proposed UML profile defines. Filters are defined as stereotypes extending the `Filter` stereotype (Figure 5). New filters can be defined as its subclasses. This section shows five filters to specify message transmission semantics and a filter to specify message processing semantics.

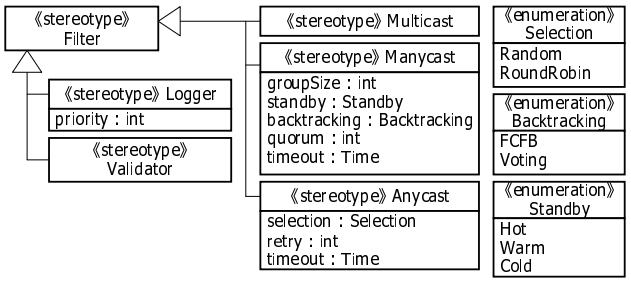

Figure 5. Tagged-Values of Filters

The stereotypes `Multicast`, `Manycast`, `Anycast`, `Logger` and `Digester` are used to define the message transmission semantics in a connector. `Multicast` receives a request message from its source and transmits it to multiple destinations simultaneously (one-to-many message exchange). When the `Multicast` filter receives reply messages from the destinations, it sends them back to the source of the request message. `Multicast` is used to improve the efficiency of message transmissions.

`Manycast` is used to improve fault tolerance by forwarding a request message to a group of replicated destinations (i.e., to the same type of services). The tagged-value `groupSize` specifies how many services are deployed as a group. `standby` specifies the operation of replicated services: *hot standby*, *warm standby* or *cold standby*. In hot standby, all services in a group remain active to receive request messages. A `Manycast` filter sends a message to all services in a group. `Manycast` returns only one reply message to the source of a request message, out of multiple replies from services. `backtracking` defines two policies to decide which reply message to be returned. When `FCFB` (first-come-first-backtracked) is selected, a `Manycast` filter returns the first reply that it receives from destination services. When `Voting` is selected, the `Manycast` filter performs a voting process. It counts the number of reply messages and inspects their contents. If the number of replies that have the same content reaches `quorum`, the `Manycast` filter returns one of the replies. If the number does not reach `quorum` within `timeout`, the `Manycast` filter returns the reply that generates the highest voting count.

In warm standby, all services in a group remain active to receive request messages. A `Manycast` filter sends a message to all services in a group, but only one service returns a reply. In this case, `backtracking` is not used. In cold standby, only one service in a group is active, and a `Manycast` filter sends a message to the service. If the service does not respond within `timeout`, the filter activates another service in the group and sends a message to the service. In cold standby, `backtracking` is not used.

An example model in Figure 6 uses a manycast filter, `Replicator`, in the connection `RecordConn`. The filter intercepts each `Inquiry` (request) message and sends it to three replicated instances of `MedicalRecordServer`, which is maintained with the hot standby policy. `Replicator` returns a `MedicalRecord` (reply message) to a `Patient` on FCFB basis.

`Anycast` is a variation of the hot standby policy in `Manycast`. It forwards a request message to only one destination in a group of replicated services. This filter is used to balance workload placed on services. `selection` defines how to choose a destination from multiple services; randomly, on round robin or on destination's priority basis (the service with the highest priority in a group is selected). If an `Anycast` filter fails to deliver a request message within `timeout`, it retries to forward the request message. `retry` specifies the maximum number of retries. If the `Anycast`

filter fails the maximum number of retries, it returns an error message to the source of the request message.

`Logger` records the transmission of each message whose priority value is higher than `priority`. When `priority` is omitted, all message transmissions are recorded. In Figure 1, the connector `RatingConn` uses a logger to record all `RatingInquiry` messages.

In addition to the filters regarding message transmission semantics, the proposed UML profile provides several other filters to specify message processing semantics in a connector. This paper describes one of them: `Validator` (Figure 5). It validates an incoming message against rules (e.g., rules specifying valid message types or data ranges), and transmits only validated messages. When a connector is encrypted with `encryptionParameter`, a `Validator` in the connector cannot validate messages. (all messages are transmitted to their destinations.)

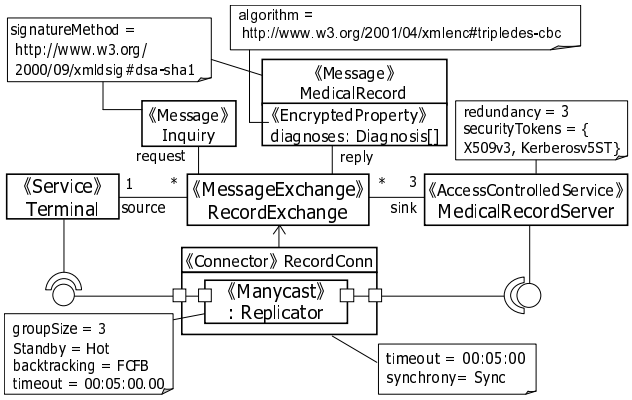

Figure 6. An Example of Manycast

3.3. Service

`Service` has three optional tagged-values (Figure 7). `priority` is the priority of each message that a service issues. Each `Anycast` filter uses `priority` to select the destination of each message, as described in Section 3.2. The range of `priority` is from 0 to 255. (0 is the lowest and 255 is the highest.)

`timeout` specifies the timeout period (in millisecond) of each message that a service issues. If a message is not delivered to its destination within this time period, a connector discards the message.

`redundancy` specifies the number of runtime instances of a service. This tagged-value must be specified when a service is accessed by `Manycast` or `Anycast` filters (Section 3.2). In Figure 6, `Replicator` (a `Manycast` filter) accesses three instances of `MedicalRecordServer` to improve the service's fault tolerance.

`AccessControlledService` is a stereotype extending `Service` (Figure 7). It represents a special type of service that enforces an access control policy. The tagged-value `securityTokens` is a mandatory tagged-value to specify security tokens (or certificates). Each `AccessControlledService` uses security tokens to authenticate the source (service) of each incoming message. This tagged-value can contain multiple values in order of precedence. The values use the names defined in the WS-SecurityPolicy specification [9]. In Figure 6, `MedicalRecordServers` control accesses from `Termials` using X.509 certificates or Kerberos tickets. X.509 certificate is used if a message sender gives both security tokens. Since UML does not provide a good means to describe policies (or rules), the proposed UML profile does not define how to specify access control policies. `AccessControlledService` is used only for indicating a service implements a certain access policy. A supporting tool transforms an `AccessControlledService` to a skeleton program code or an access control description in accordance with an implementation technology that an application developer chooses. Application developers are required to complete implementing access control policies.

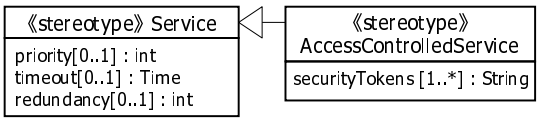

Figure 7. Tagged-Values of Service

3.4. Message

`Message` has a mandatory tagged-value, `schemaURI`, and three optional tagged-values: `priority`, `timeout` and `signatureMethod` (Figure 8). `schemaURI` identifies the schema of a message. The default value of `schemaURI` is message's qualified name (a combination of a package name and message's name).

`priority` and `timeout` specifies the priority and timeout period of messages. `Connector` and `Service` also have those tagged-values. The precedence is that `Message`'s tagged-values override `Service`'s ones, and `Service`'s tagged-values override `Connector`'s ones.

In order to ensure the integrity of a message, `signatureMethod` is used to specify an algorithm for generating the message's degital signature. The algorithm is represented with a URI defined in the XML Signature specification [10]. For example, DSA (Digital Signature Algorithm) is represented with http://www.w3.org/2000/09/xmldsig#dsa-sha1. In Figure 6, each `Inquiry` and `MedicalRecord` message is signed with DSA. When `signatureMethod` is specified, each message is expected to maintain its signature in a data field called `signature`.

The stereotype `EncryptedProperty` is used for message-level (end-to-end) encryption (see Section 3.1 for transport-level (point-to-point) encryption). It is defined

as a stereotype extending `Property` in the UML metamodel (Figure 8). This stereotype is attached to data fields to be encrypted in a message. For example, in Figure 1, `EncryptedProperty` is attached to the `SSN` data filed of the `RatingInquiry` message and the `creditRating` data filed of the `Rating` message. `EncryptedProperty` has a tagged-value, `algorithm`, to specify an algorithm used to encrypt a message. The semantics of this tagged-value is same as that of `encryptionAlgorithm` in `Connector` (Section 3.1). An encryption algorithm is specified as a URI that the XML Encryption specification defines [8]. In Figure 1, `SSN` in `Rating` is encrypted with Triple DES. Different data fields in a message can be encrypted with different encryption algorithms.

`AccessControlledMessage` is a stereotype extending `Message` (Figure 8). Similar to `AccessControlledService`, it is a special type of message that enforces an access control policy. The tagged-value `securityTokens` is mandatory to specify security tokens (or certificates). The security tokens are used to authenticate entities (e.g., services) that access a message. This tagged-value can contain multiple values in order of precedence. The values use the names defined in the WS-SecurityPolicy specification [9]. Since UML does not provide a good means to describe policies (or rules), the proposed UML profile does not define how to specify access control policies. `AccessControlledMessage` is used to indicate a message implements a certain access policy. A supporting tool transforms an `AccessControlledMessage` to a skeleton program code or an access control description in accordance with an implementation technology that an application developer chooses. Application developers are required to complete implementing access control policies.

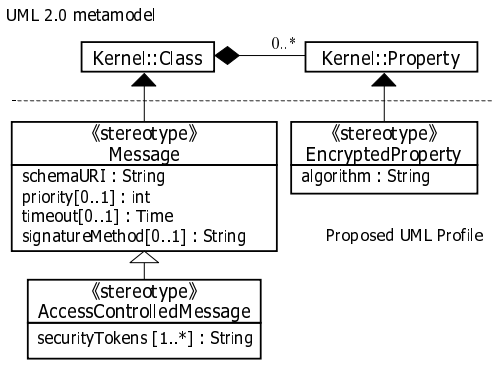

Figure 8. Tagged-Values of Message

4. Secure Application Development with the Proposed UML Profile

This section describes a model-driven development (MDD) tool, called Ark, which accepts a UML model designed with the proposed profile and transforms the model into a skeleton of application code (source code and deployment descriptor).

Currently, Ark implements a transformation mapping between the proposed UML profile and MuleESB[2]. Ark takes a UML model in the XML Metadata Interchange (XMI) format. It has been tested with MagicDraw[3], a visual UML modeling tool that can serialize UML models to XMI. An input UML model (XMI file) is validated against the UML metamodel and the proposed profile, and transformed to Java programs and deployment descriptors for MuleESB. A mapping rule between the proposed profile and MuleESB is implemented as a set of Velocity[4] transformation templates, which define how to transform UML model elements to application code elements.

Figure 9 shows some of the Java classes and deployment descriptors that Ark generates from the UML model in Figure 1. Ark maps a UML class stereotyped with ≪`Message`≫ to a Java class that has the same class name and the same data fields (Figure 9). The Java class implements the interface `Serializable`. This is required to implement messages exchanged with MuleESB.

A UML class stereotyped with ≪`Service`≫ is mapped to a Java class that has the same class name and the same data fields. Ark inserts several operations to the Java class, depending on whether its association role is `source` or `sink` against a message exchange. The operations are used to send and receive messages: `sendX()` to send messages where `X` references the name of a message exchange, and `onMessage()` to receive messages. For example, in Figure 9, `Bank` has `sendLoanAppProcessing()` to send `LoanApproval` messages and `onMessage()` to receive `LoanApp` messages.

UML classes stereotyped with ≪`MessageExchange`≫ and ≪`Connector`≫ are not mapped to particular Java classes. The message transmission/processing semantics specified in a `Connector` is implemented in the Java classes of message source and destination. For example, in Figure 1, an `Appricant` sends each `LoanApp` message to a `Bank` synchronously. Therefore, Ark generates a code fragment to send the message synchronously using MuleESB's API[5], and embeds the code in `sendLoanAppProcessing()` of `Applicant`. Ark also generates a code fragment to handle timeout using MuleESB's API, and embeds the code in `sendLoanAppProcessing()` of `Applicant`.

As Figure 1 shows, `SSN` in the `RatingInquiry` message and `creditRating` in the `Rating` message are en-

[2] A major open-source ESB implementation. http://mule.codehaus.org/

[3] http://www.magicdraw.com/

[4] A template-based code generation engine. jakarta.apache.org/velocity

[5] MuleESB provides three different APIs to send messages in synchronous, asynchronous and oneway (non-blocking) manner.

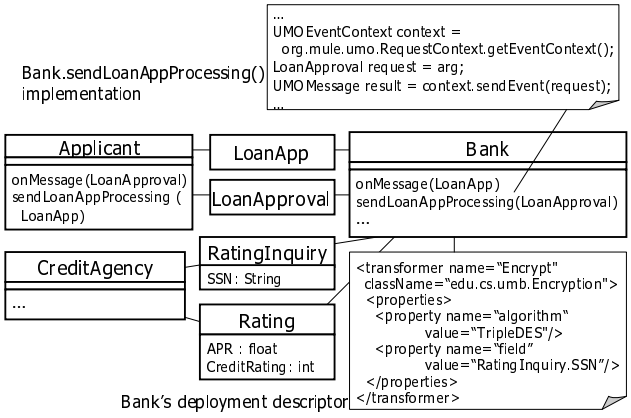

Figure 9. Generated Code for MuleESB

crypted. Since MuleESB does not support message-level encryption, Ark provides a pair of message transformers to encrypt and decrypt data fields in messages. In MuleESB, each service can have an arbitrary number of message transformers as the classes implementing the interface `org.mule.transformer.UMOTransformer`. Message transformers are invoked when a service receives a message or when it sends out a message. Ark generates a deployment descriptor to configure services that send or receive encrypted messages so that the services use the message encryption/decryption transformers in Ark. Figure 9 shows a fragment of generated deployment descriptor for `Bank`. It configures `Bank` to use a message encryption transformer (`edu.cs.umb.Encryption`) to encrypt the data field `SSN` in `RatingInquiry` using Triple DES.

5. Related Work

There are several UML profiles proposed for SOA. [11] and [12] propose UML profiles to specify functional aspects in SOA. Both profiles are defined based on the XML schema of Web Service Description Language (WSDL). Each of the profiles provides a set of stereotypes and tagged-values that correspond to elements in WSDL, such as `Service`, `Port`, `Messages` and `Binding`[6]. Since WSDL is designed to define only functional aspects of web services, non-functional aspects are beyond of the scope of [11] and [12]. The proposed profile focuses on specifying non-functional aspects in SOA.

[13] proposes a UML profile to describe both functional and non-functional aspects in SOA. The stereotypes in this profile are generic enough to specify a wide range of applications. However, their semantics tend to be ambiguous. For example, the stereotypes for non-functional aspects include ≪`policy`≫, ≪`permission`≫ and ≪`obligation`≫, and ≪obligation≫ is intended to specify the responsibility of a service. [13] does not precisely define what developers have to (or can) specify with this stereotype and how to represent service responsibility (e.g., using natural languages or parameter values). In contrast, the proposed profile carefully defines its stereotypes and tagged-values in an unambiguous manner so that supporting tools can interpret and transform models to code.

[14] describes a UML profile for data integration in SOA. It provides data structures to specify messages so that users can build data dictionaries that maintain message data used in existing systems and new applications. This profile separates a non-functional aspect in data integration from functional aspects, and enables data integration in an implementation independent manner. The proposed profile focuses on non-functional semantics in message transmission, message processing, security and service deployment (e.g., service redundancy), rather than data integration.

[15] proposes a UML profile to facilitate service discovery in SOA. This profile provides a set of stereotypes (e.g., ≪`uses`≫, ≪`requires`≫ and ≪`satisfies`≫) to specify relationships among service implementations, service interfaces and functional requirements. For examples, users can specify relationships in which a service *uses* other services, and a service *requires* other services that *satisfy* certain functional requirements. These relationship specifications are intended to effectively aid dynamic discovery of services. The proposed profile and [15] focus on different issues in SOA. Service discovery is beyond of the scope of the proposed profile, and [15] does not consider non-functional aspects in message transmission, message processing, security and service deployment.

[16], [17] and [18] define UML profiles to specify service orchestration and map it to Business Process Execution Language (BPEL) [19]. These profiles provide a limited support of non-functional aspects in message transmission, such as messaging synchrony. The proposed profile does not focus on service orchestration, but a comprehensive support of non-functional aspects in message transmission, message processing, security and service deployment.

[20] proposes a UML profile, called SecureUML, to define role-based access control for network applications. SecureUML provides the notations to assign roles (≪`security.role`≫) and access control permissions (≪`security.constraint`≫) to classes. SecureUML employs Object Constraint Language (OCL) to define access control. [21] defines another UML profile, called UMLsec, to define data encryption (≪`data security`≫) and secure network links (≪`encrypted`≫). [22, 23, 24] also propose UML profiles to define security aspects. These UML profiles are parallel to the proposed profile in terms of the ability to describe security aspects in network applications. However, the proposed UML profile allows users

[6] In WSDL, `Service` defines an interface of a web service. `Port` specifies an operation in a `Service`, and `Message` defines parameters of a `Port`. `Binding` specifies communication protocols used by `Ports`.

to consistently specify many other non-functional aspects (message transmission, message processing and service deployment) as well as security aspects in SOA.

There are several specifications and research efforts to investigate implementation techniques for non-functional aspects in SOA [19, 25, 26, 27, 28, 29]. Each specification and technique provides a means to implement non-functional requirements in, for example, performance, reliability and security and to enforce services to follow the requirements. Rather than providing specific implementations of non-functional aspects in SOA, the proposed UML profile is intended to provide a means for users to model and maintain non-functional aspects in an implementation independent manner so that they can be mapped on different specifications or implementation technologies.

6. Concluding Remarks

This paper proposes a UML profile to graphically specify and maintain non-functional aspects in SOA (particularly security aspect) in an implementation independent manner. This paper presents design details of the proposed profile, and describes how MDD tools can use it to develop secure service-oriented applications. As an example of MDD tools, this paper demonstrates a tool that accepts a UML model defined with the proposed profile and transforms it into application code for Mule ESB and GridFTP.

7. Acknowledgement

This work is supported in part by OGIS International, Inc. and Electric Power Development Co., Ltd.

References

[1] S. Vinoski. Integration with Web Services. *IEEE Internet Computing*, November/December 2003.

[2] Z. Zhang and H. Yang. Incubating Services in Legacy Systems for Architectural Migration. *Asia-Pacific Software Engineering Conference*, December 2004.

[3] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, and T. Newling. *Patterns: Service-Oriented Architecture and Web Services*. IBM Red Books, 2004.

[4] G. Lewis, E. Morris, L. Brien, D. Smith, and L. Wrage. SMART: The Service-Oriented Migration and Reuse Technique. Technical report, Software Engineering Institute, Carnegie Mellon University, September 2005.

[5] Object Management Group. UML2.0 Super Structure Specification, October 2004.

[6] L. Fuentes and A. Vallecillo. An Introduction to UML Profiles. *The European journal for the Informatics Professional*, April 2004.

[7] D. Chappell. *Enterprise Service Bus*. O'Reilly, June 2004.

[8] The World Wide Web Comsortium. XML Encryption Syntax and Processing, December 2002.

[9] RSA Security IBM, Microsoft and VeriSign. Web Services Security Policy Language, December 2002.

[10] The World Wide Web Comsortium. XML Signature Syntax and Processing, February 2002.

[11] E. Marcos, V. de Castro, and B. Vela. Representing Web services with UML: A Case Study. *the Int'l Conference on Service Oriented Computing*, December 2003.

[12] IBM. *UML 2.0 Profile for Software Services*. developerWorks, April 2005.

[13] R. Amir and A. Zeid. A UML Profile for Service Oriented Architectures. *ACM OOPSLA Poster session*, 2004.

[14] M. Vokäc. Using a Domain-Specific Language and Custom Tools to Model a Multi-tier Service-Oriented Application–experiences and challenges. *ACM/IEEE Int'l Conference on Model Driven Engineering Languages and Systems*, October 2005.

[15] R. Heckel, M. Lohmann, and S. Thöne. Towards a UML Profile for Service-Oriented Architectures. *Workshop on Model Driven Architecture: Foundations and Applications*, 2003.

[16] T. Gardner. UML Modeling of Automated Business Processes with a Mapping to BPEL4WS. *ECOOP Workshop on OO and Web Services*, July 2003.

[17] IBM. *UML 1.4 Profile for Software Services with a Mapping to BPEL 1.0*. developerWorks, July 2004.

[18] Object Management Group. Business Process Definition Metamodel, January 2003.

[19] OASIS. Web Services Business Process Execution Language, April 2003.

[20] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. *ACM/IEEE Int'l Conference on Unified Modeling Language*, October 2002.

[21] J. Jŕjens. UMLsec: Extending UML for Secure Systems Development. *ACM/IEEE Int'l Conference on Unified Modeling Language*, October 2002.

[22] L. Wang and L. Lee. UML-based Modeling of Web Services Security. *IEEE European Conference on Web Services Poster session*, 2005.

[23] Y. Nakamura, M. Tatsubori, T. Imamura, and K. Ono. Model-Driven Security Based on a Web Services Security Architecture. *IEEE Int'l Conference on Services Computing*, July 2005.

[24] R. Villarroel, E. Medina, M. Piattini, and J. Trujillo. A UML 2.0/OCL Extension for Designing Secure Data Warehouses. *Journal of Research and Practice in Information Technology*, February 2006.

[25] OASIS. Web Service Reliable Messaging, September 2004.

[26] OASIS. Web Service Reliability 1.1, November 2004.

[27] F. Baligand and V. Monfort. A Concrete Solution for Web Services Adaptability Using Policies and Aspects. *Int'l Conf. on Service Oriented Computing*, December 2004.

[28] G. Wang, A. Chen, C. Wang, C. Fung, and S. Uczekaj. Integrated Quality of Service (QoS) Management in Service-Oriented Enterprise Architectures. *IEEE Enterprise Distributed Object Computing Conference*, 2004.

[29] N. Mukhi, R. Konuru, and F. Curbera. Cooperative Middleware Specialization for Service Oriented Architectures. *ACM Int'l World Wide Web Conference*, 2004.