International Journal of Web Services Research , Vol.X, No.X, 200X

A Model-Driven Development Framework for Non-Functional Aspects in Service Oriented Architecture

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.
San Mateo, CA 94404
oba@ogis-international.com

ABSTRACT:

Service Oriented Architecture (SOA) is an emerging style of software architectures 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, 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 proposes a model-driven development (MDD) framework for non-functional aspects in SOA. The proposed MDD framework consists of (1) a Unified Modeling Language (UML) profile to graphically model non-functional aspects in SOA, and (2) an MDD tool that accepts a UML model defined with the proposed profile and transforms it to application code. This paper also demonstrates how the proposed framework is used in model-driven development of service-oriented applications. Empirical evaluation results show that the proposed MDD framework improves the reusability and maintainability of service-oriented applications by hiding low-level implementation technologies in UML models.

KEY WORDS:
Service Oriented Architecture, Visual Non-functional Modeling, UML, Metamodeling, Model Driven Development

INTRODUCTION

A key challenge in large-scale distributed systems is to reuse and integrate existing systems to build new applications in a cost effective manner (Vinoski, 2003; Zhang, 2004). Service Oriented Architecture (SOA) addresses this challenge by improving the reusability and maintainability of distributed systems (Papazoglou, 2003; Endrei et al., 2004; Lewis et al., 2005; Foster, 2005; Bichler et al., 2006; Arsanjani et al., 2007). It is an emerging style of software architectures 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 languages and remoting middleware) from application developers. They can reuse and combine services to build their applications without knowing the implementation details of services and connections.

In order to make this vision of SOA a reality, this paper focuses on a research issue of increasing the reusability of services and connections, and addresses this issue by separating non-functional aspects (e.g., security and fault tolerance) of services and connections from their functional aspects. The separation of functional and non-functional aspects can improve the reusability of

services and connections because it allows different applications to use services and connections in different non-functional contexts. For example, an application may unicast messages to a service, and another may manycast messages to multiple replicas of the service to improve fault tolerance. Also, an application may send signed and encrypted messages to a service, when the messages travel to the 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 can also improve the ease of understanding application design and enable the two different aspects to evolve independently. This results in higher maintainability of applications.

This paper describes a model-driven development (MDD) framework for non-functional aspects in SOA. The MDD framework consists of (1) a Unified Modeling Language (UML) profile to model non-functional aspects in SOA, and (2) an MDD tool that accepts a UML model defined with the proposed profile and transforms it to application code (e.g., program code and deployment descriptors). The proposed UML profile allows application developers to graphically describe and maintain non-functional aspects in SOA as UML diagrams (composite structure diagrams and class diagrams). Using the proposed UML profile, non-functional aspects can be modeled without depending on any particular implementation technologies. The proposed MDD tool, called Ark, transforms implementation independent UML models into implementation specific application code.

This paper describes design details of the proposed UML profile, and demonstrates how Ark transforms an input UML model to application code that runs with certain implementation technologies such as Enterprise Service Buses (ESBs) (Chappell, 2004), secure file transfer protocols and grid computing platforms. Empirical evaluation results show that the proposed MDD framework improves the reusability and maintainability of service-oriented applications by hiding implementation technologies in UML models.

CONTRIBUTIONS

This paper offers the following three contributions to the design space of service-oriented applications.

- **Modeling Support for Non-functional Aspects in SOA**: This work is the first attempt to investigate a UML profile to consistently model a wide range of non-functional aspects in SOA, although there exist several UML profiles for specific aspects (e.g., functional aspects and service discovery) in SOA. (See the Related Work section for more details.) The proposed UML profile covers the following four areas of non-functional aspects.

    1. *Service Deployment Semantics*: Service redundancy.
    2. *Message Transmission Semantics*: Messaging synchrony, message delivery assurance, message queuing, multicast, manycast, anycast, message routing, message prioritization, messaging timeout, message logging, and message retention.
    3. *Message Processing Semantics*: Message conversion, message split, message aggregation, message validation, and message filtering.
    4. *Security Semantics*: Transport-level encryption, message-level encryption (entire/partial message encryption), message signature, message access control, and service access control.

- **Modeling Support for Regulatory Compliance**: As regulatory compliance has been becoming an important factor in software development and maintenance, regulatory mandates (e.g., the Sarbanes-Oxley Act and HIPPA) dramatically increase the number of non-functional aspects that application developers need to consider (O'Grady, 2004). This work is the first attempt to investigate a visual modeling language to describe non-functional aspects derived from regulatory mandates. The proposed UML profile allows application developers (or compliance management staffs) to graphically specify and verify how their applications meet regulatory mandates. Currently, the proposed UML profile addresses data retention, data/process validation (e.g., consistency validation among an order, invoice and payment) and security (e.g., access control and data integrity).

- **MDD Support for Service-Oriented Applications**: Non-functional requirements change during application lifecycle more often than functional aspects (Bieberstein et al., 2005). It can be expensive to manage frequent changes in non-functional requirements. This results in escalating maintenance cost, in turn total cost of owning. When a non-functional requirement (e.g., security policy) changes in an application, the proposed MDD framework allows application developers to make the change in a UML model specifying the application's non-functional aspects and keep its functional part intact. The proposed MDD tool (Ark) generates non-functional code from the updated UML model and combines the generated code with existing functional code. Ark makes application's functional aspects reusable across the changes in non-functional requirements, thereby improving the productivity of application development and maintenance.

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 (Object Management Group, 2004), which is the grammar specification for standard (default) model elements in UML.

In addition to standard model elements, UML provides extension mechanisms (e.g., stereotypes and tagged-values) to specialize them to precisely describe domain or application specific concepts (Fuentes et al., 2004). 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.

For example, a UML profile for Enterprise Java Beans (EJB) (Java Community Process, 2001) defines the stereotype `<<EJBEntityBean>>`, which extends `Class` in the UML metamodel. This means the stereotype can be applied to classes. Thus, a UML class stereotyped with `<<EJBEntityBean>>` indicates that the class is designed as an EJB entity bean. The stereotype `<<EJBEntityBean>>` has a tagged-value, called `EJBPersisitenceType`, to specify who provides persistence to an entity bean. The tagged-value can have a value `Bean` or `Container`. `Bean` indicates an individual entity bean is responsible for its own persistence, and `Container` indicates an EJB container takes care of persistence.

Figure 1 overviews an example purchasing system across buyers, retailers, suppliers and inventory managers. All example models in this paper focus on and define several particular parts

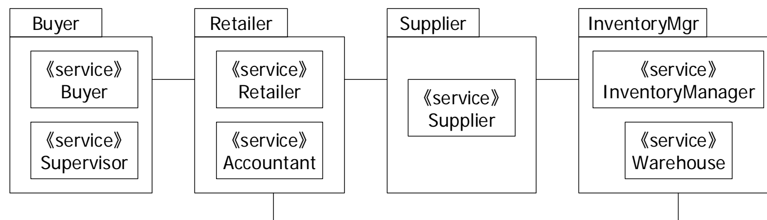

Figure 1. The Structural Architecture of an Example Purchasing System

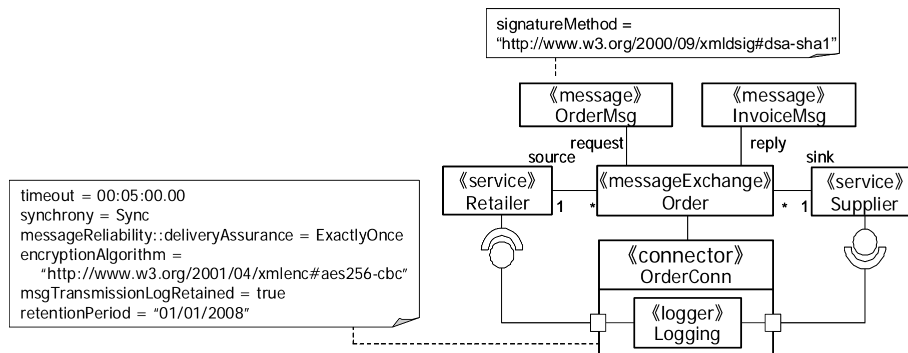

Figure 2. An Example UML Model

of this system. In this example system, a `Buyer` purchases a product from a `Supplier` via `Retailer`. A `Supervisor` authorizes each order that a `Buyer` places. An `Accountant` performs accounting tasks for a `Retailer`. An `InventoryManager` manages a `Retailer`'s inventory.

Figure 2 shows an example model built with the proposed UML profile. This model focuses on an interaction between a `Retailer` and `Supplier` in Figure 1, and defines an order processing scenario in which a `Retailer` places an order and a `Supplier` issues an invoice. In this example, two services (`Retailer` and `Supplier`) exchange messages. Each service is represented by a class stereotyped with `<<service>>`. These services exchange two types of messages (`OrderMsg` and `InvoiceMsg`), each of which is stereotyped with `<<message>>`. Each message can have multiple tagged-values to specify additional message transmission/processing semantics. In this example, the tagged-value `signatureMethod` specifies that an `OrderMsg` carries a digital signature created with DSA (Digital Signature Algorithm). Each pair of a request and reply messages is represented by a class stereotyped with `<<messageExchange>>`.

`<<connector>>` represents a connection that transmits messages between services. In this example, messages are delivered through a connector called `OrderConn`. Every message exchange is bound with a connector in order to specify which connector is used to deliver messages. A connector has a provided interface (represented as a "ball" notation) and a required interface (represented as a "socket" notation) to transmit messages between services. Services use the provided and required interfaces to send and receive messages, respectively. The two interfaces are intended to show how services use (connect with) a connector.

Each connector can have multiple filters inside. They are used to define message transmission/processing semantics in a connector. This example uses a `Logger` in the `OrderConn` connector. `Logger` logs messages that transmitted through the filter (`OrderMsg` and `InvoiceMsg` in this example).

Also, each connector can have multiple tagged-values to specify additional message transmission/processing semantics. In this example, `OrderConn` specifies the timeout of message transmissions (five minutes), the synchrony of message transmissions (synchronous), the assurance level of message delivery (exactly once) and the message encryption algorithm (Advanced Encryption Standard). Also, through the use of tagged-values `msgTransmissionLogRetained` and `retentionPeriod`, `OrderConn` specifies to retain the logs of message transmissions until a certain date.

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.

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). Figure 3 shows how the proposed profile defines these stereotypes by extending the UML metamodel. Each stereotype is defined as a metaclass stereotyped with `<<stereotype>>`[1]. 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 2). 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` (Figure 2). Each `MessageExchange` may have multiple reply messages per request message (Figure 3). 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 2 shows a

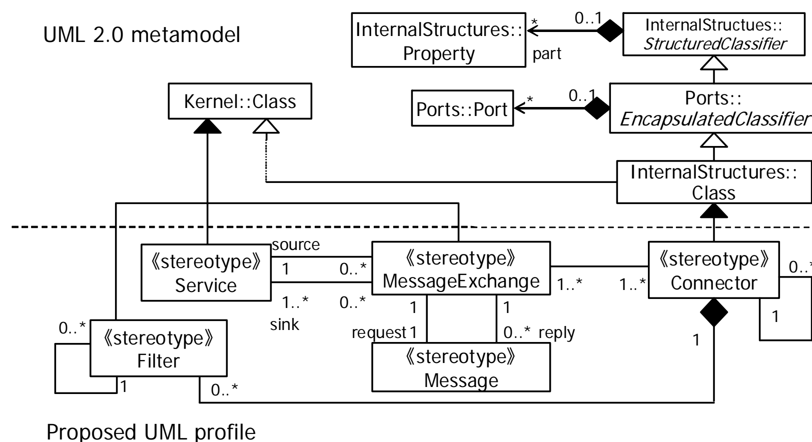

Figure 3. Definition of Stereotypes

[1] According to the UML specification, the first letter of a stereotype's name is capitalized when the stereotype is defined (Figure 3). However, it is not capitalized when the stereotype is used in UML models (Figure 2).

one-to-one message exchange between a Retailer and a Supplier.

Connector is a stereotype extending the Class metaclass in the InternalStructures package of the UML metamodel (Figure 3). This metaclass defines a composite class, a special type of class, which can contain other model elements (e.g., inner classes)[2] 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 2 shows the OrderConn connector , which contains a filter (a Logger). This filter receives, records message's log, and sends out OrderMsg or InvoiceMsg messages.

Table 1. Key Model Elements (Stereotypes) in the Proposed UML Profile

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>>	Customizes the semantics of message transmission and message processing in a connector.

Connector

Connector has 10 tagged-values (Figure 4). 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 2, the timeout period of the connector OrderConn is specified as five 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 4), and each connector chooses one of them. In Figure 2, a Retailer and a Supplier exchange OrderMsg and InvoiceMsg messages synchronously.

priority is a mandatory tagged-value to specify the priority of each message that a connector delivers. The range of priority is from 0 to 255 (0 is the lowest and 255 is the highest), and the default value is 0 (Figure 4).

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.

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

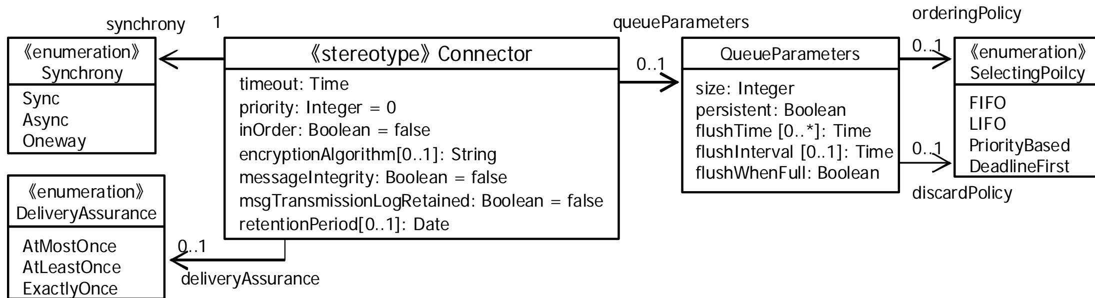

Figure 4. Tagged-Values of `Connector`

`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 4), and each `Connector` chooses one of them at a time. `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 5 shows an example model using `inOrder` and `deliveryAssurance`. This example illustrates an extension to an order processing application in Figure 2. In this example, a `Buyer` transmits an `OrderMsg` to a `Supplier` via `Retailer` (See also Figure 1.) 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 a `Supplier` before a corresponding order cancellation. Therefore, the `inOrder` semantics is assigned to the `OrderConn` connector. This semantics implicitly assigns `ExactlyOnce` to the `deliveryAssuarance` semantics in the `OrderConn` connector.

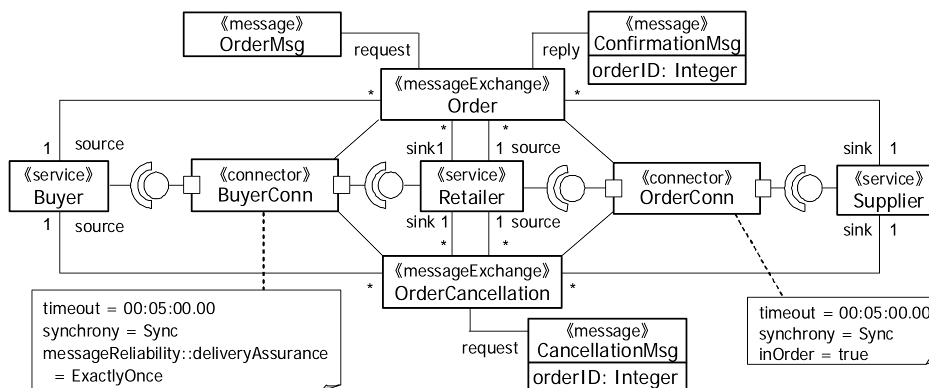

Figure 5. An Example of `inOrder` and `deliveryAssurance`

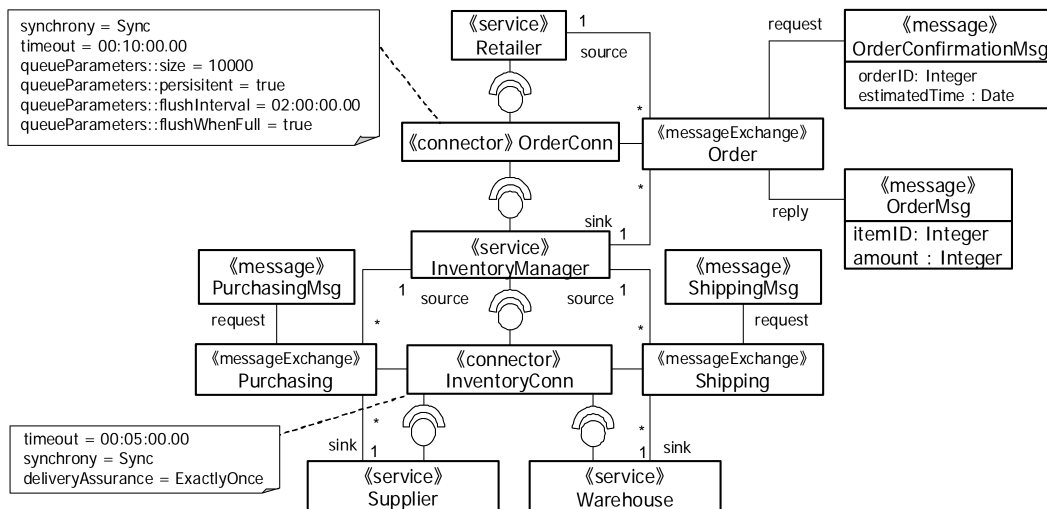

Figure 6. An Example of `Queue`

`encryptionAlgorithm` is an optional tagged-value used for transport-level encryption in a connector. This tagged-value defines an algorithm to secure a connection upon which request and response messages are transmitted. (See Figure 2 for an example.) The encryption algorithm is specified as a URI defined in the XML Encryption specification (World Wide Web Consortium, 2002). 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`.

`queueParameters` is an optional tagged-value to deploy a message queue between services (i.e., message source and destination) and specify the semantics of message queuing between them. `size` specifies the maximum number of queued messages. `flushWhenFull` specifies whether queued messages are flushed from a queue to their destinations when the queue overflows. When `flushWhenFull` is false, the overflowing queue discards a message according to `discardPolicy` (Figure 4); discarding the oldest message (First-In-First-Out), the newest message (Last-In-First-Out), the lowest priority message or the closest deadline message. These four policies are defined as an enumeration in `SelectionPolicy` (Figure 4). `flushTime` and `flushInterval` specify when and how often a queue flushes messages, respectively. `orderingPolicy` specifies how to order messages in a queue: FIFO, LIFO, highest-priority-first or earliest-deadline-first. `persistent` specifies whether a queue stores messages in a storage (e.g., a file or database) so that the queue can recover them when it crashes unexpectedly.

Figure 6 shows an example using `queueParameters`. It illustrates an inventory management application for retailers. (See also Figure 1) Each `Retailer` transmits an `OrderMsg` to an `InventoryManager` when it has no or few products in stock. The `InventoryManager` receives `OrderMsgs` from multiple `Retailers` every two hours in a batch manner. The `OrderConn` connector implements a synchronous queue that stores and forwards `OrderMsgs`. The `InventoryManager` schedules which warehouses deliver which products to which retail stores (every two hours), and based on the shipping schedule, sends `ShippingMsgs` to

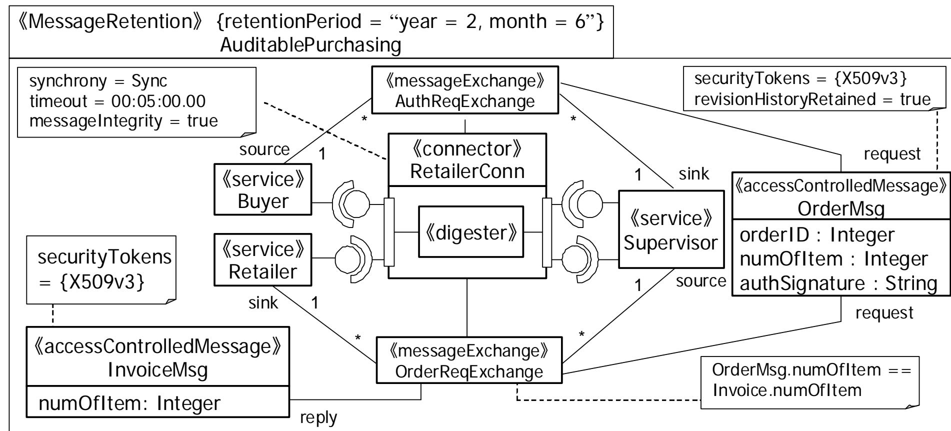

Figure 7. An Example Model for Regulatory Compliance

Warehouses. If a warehouse has a small inventory of a particular product, the InventoryManager orders the product by sending a PurchasingMsg to a Supplier.

msgTransmissionLogRetained is a mandatory tagged-value to specify whether to retain logs on message transmissions. (See Figure 2 for an example.) Regulatory mandates require applications to retain the logs and make them auditable for the third party organizations in the future. A connector with this tagged-value records (1) which messages are transmitted, (2) message source and destination (services), and (3) when the messages are transmitted. If msgTransmissionLogRetained is true, retentionPeriod must be specified to define the period to retain each message transmission log. The default value of msgTransmissionLogRetained is false. If it is false, retentionPeriod is ignored.

messageIntegrity is a mandatory tagged-value to specify whether to ensure the message integrity. The default value of messageIntegrity is false. A connector with this tagged-value checks whether messages are changed during their transmission.

Figure 7 illustrates an order processing application in which a Buyer places an order and a Retailer receives it via authorization by a Supervisor. By assigning a signature to the authSignature data field of an OrderMsg, Supervisor authorizes the message (order). Services are connected through a connector with the messageIntegrity semantics. This semantics ensures that OrderMsg messages are not altered during their transmission, and eliminates the possibility of malicious alteration.

A package stereotyped with <<messageRetention>> specifies that contained connectors have the msgTransmissionLogRetained semantics implicitly if the connectors omit it (Figure 7). Each connector follows the retentionPeriod specified in the package. When a connector specifies msgTransmissionLogRetained and retentionPeriod explicitly, they override the retentionPeriod specified in a package. Also, a package stereotyped with <<messageRetention>> enforces contained services and messages to log their message transmissions. Connectors, services and messages retain their logs independently, and the third party organizations can discover fraud activities by checking the inconsistencies between their logs.

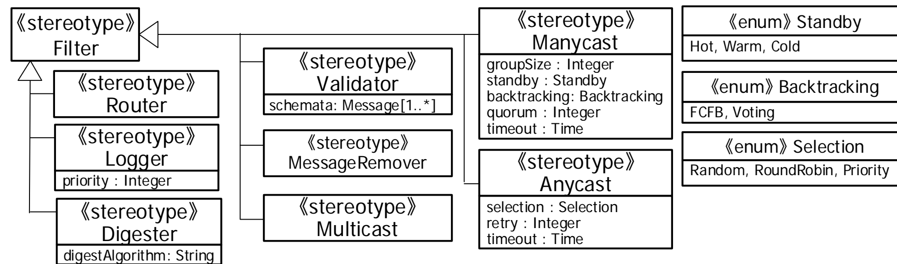

Figure 8. Tagged-Values of Filters

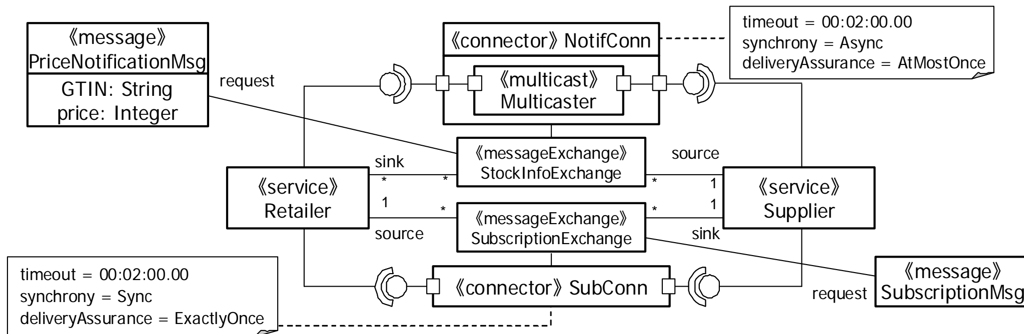

Figure 9. An Example of `Multicast`

Application developers can specify constraints using the Object Constraint Language (OCL). Constraints are used to ensure the consistency among application data and check whether services work correctly. For example, in Figure 7, `OrderReqExchange` has a constraint that ensures the number of items in an order (`OrderMsg`) and a corresponding invoice (`InvoiceMsg`) are always equal. When this constraint is violated, fraud activities could be committed.

Filter

This paper describes eight of the filters that the proposed UML profile defines. Filters are defined as stereotypes extending the `Filter` stereotype (Figure 8). New filters can be defined as its subclasses. This section shows six filters to specify message transmission semantics and two filters to specify message processing semantics.
The stereotypes `Multicast`, `Manycast`, `Anycast`, `Router`, `Logger` and `Digester` are used to define the message transmission semantics in a connector. A `Multicast` filter receives a request message from its source and transmits it to multiple destinations (services) simultaneously (one-to-many message exchange). A group of destinations can contain different types of services. 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.

Figure 9 shows an example that models an application for wholesale price notification using `Multicast`. A `Retailer` subscribes for the price changes of a particular supply, and a `Supplier` notifies (or publishes) any price changes to the `Retailer`. A `Retailer` transmits a `Subscription` message to a `Supplier` in a synchronous and exactly-once manner. A `Supplier` multicasts a `PriceNotificationMsg` message, which contains a supply's GTIN

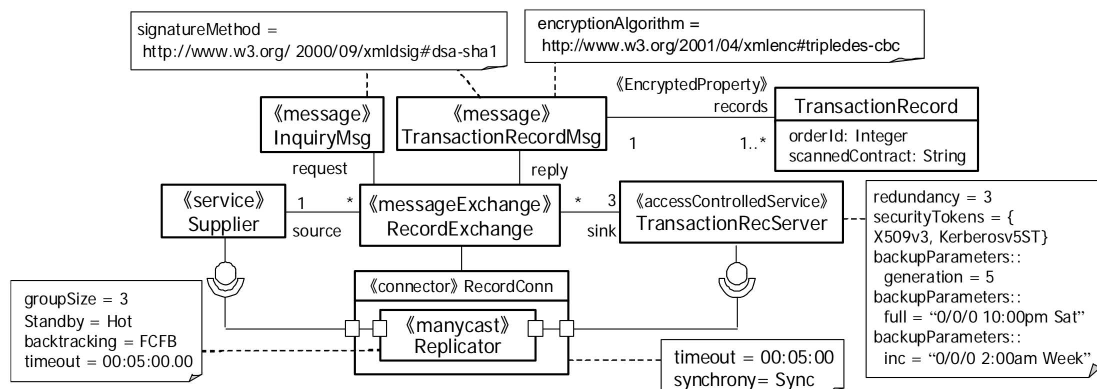

Figure 10. An Example of `Manycast`

(Global Trade Item Number)[3] and price, to multiple `Retailers` in asynchronous and at-most-once semantics.

`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.

In an example model shown in Figure 10, a supplier sends an inquiry to a cluster of transaction record servers to obtain a transaction record containing a set of orders. A manycast filter, `Replicator`, is used in the connection `RecordConn`. The filter intercepts each `Inquiry` (request) message and sends it to three replicated instances of `TransactionRecServer`, which is maintained with the hot standby policy. `Replicator` returns a `TransactionRecordMsg` (reply message) to a `Supplier` on FCFB basis.

[3] http://www.gtin.info/

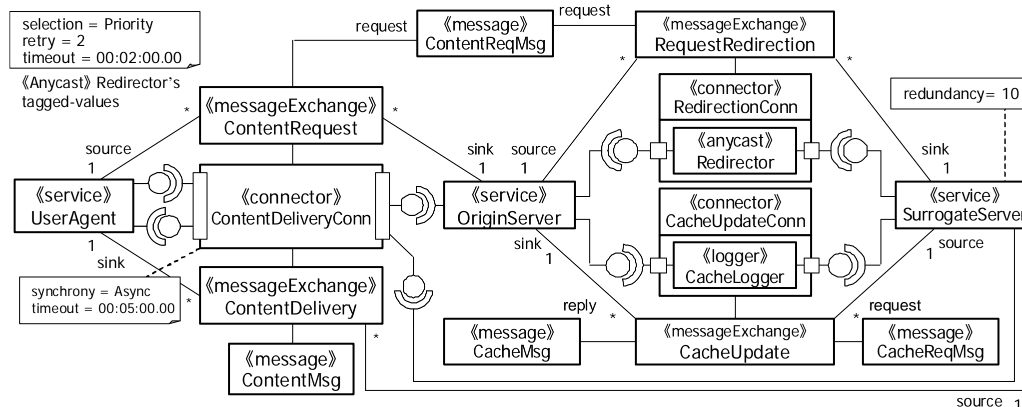

Figure 11. An Example of `Anycast`

`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.

Figure 11 shows an example model describing a content delivery system, e.g., for delivering contents among supplier's on-line catalogs of their supplies. (For simplicity, tagged-values of the connector `RedirectionConn` and `CacheUpdateConn` are omitted, but both have the synchronous semantics and their timeout is five minutes.) A user agent (`UserAgent`) sends a request (`ContentReqMsg`) to a content server (`OriginServer`) through a connector (`ContentDeliveryConn`). To balance workload, the content server redirects the request to a surrogate server (`SurrogateServer`). This model has a cluster of surrogate servers which consists of 10 replica servers. An `Anycast` filter in the connector `RedirectionConn` selects one of servers on their priority basis, and redirects a request to it. (tagged-values of the `Anycast` filter is described on the upper left corner.) If a surrogate server does not have data to process a request, it sends a request (`CacheReqMsg`) to a content server to obtain cache data (`CacheMsg`). After processing a request, a surrogate server returns content (`ContentMsg`) to an user agent.

`Router` routes an incoming message to one or more destinations with certain criteria. Since UML does not provide a means to define rules, the proposed profile has no facility to specify routing rules at design time. Supporting tools transform a `Router` to a skeleton source code (e.g., in Java) or a rule description (e.g., in XPath) that performs message routing. Developers are expected to complete the skeleton code/description.

`Logger` records the transmission of each message whose priority value is higher than `priority`. When `priority` is omitted, all message transmissions are recorded.

`Digester` records digest values of all messages. This filter can be used to check whether a message is altered after its transmission. The digest algorithm is specified as a URI defined in the

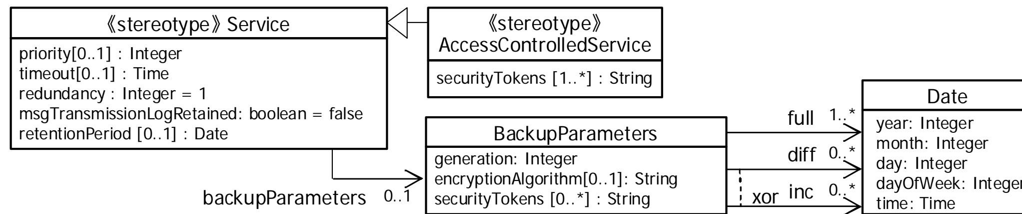

Figure 12. Tagged-Value of `Service`

XML Encryption specification (The World Wide Web Consortium, 2002a). For example, `http://www.w3.org/2000/09/xmldsig#sha1` specifies SHA-1.

In addition to the stereotypes for message transmission semantics, the proposed UML profile provides two other stereotypes to define the message processing semantics in each connector: `Validator` and `MessageRemover` (Figure 8). `Validator` and `MessageRemover` validate incoming messages against the message schema specified in its tagged-value `schemaURI` and a given criteria (e.g., rules specifying valid data ranges) respectively, and transmit only validated messages. Since UML does not provide a means to define rules, the proposed profile has no facility to specify message filtering rules for `MessageRemover` at design time. Supporting tools transform a `Validator` and `MessageRemover` to a skeleton source code (e.g., in Java) or a rule description (e.g., in XPath) that performs message filtering. Developers are expected to complete the skeleton code/description. When a connector is encrypted with `encryptionParameter`, `Validator` and `MessageRemover` in the connector cannot validate messages (all messages are transmitted to their destinations.)

Service

`Service` has six tagged-values (Figure 12). `timeout` is an optional tagged-value to specify 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.

`priority` is an optional tagged-value to specify the priority of each message that a service issues. `Anycast` filter uses `priority` to select its destinations. Also, it is used to order messages in a message queue when a connector has `queueParameters`.

Each service is expected to have data fields corresponding to the `priority` and `timeout` tagged-values. Usually, class instances cannot read and write tagged-values because tagged-values are defined in a metamodel (see Figure 3) and used in a model. The data fields allow different class instances to have different tagged-values, and tagged-values specified in a model behave as default values of corresponding data fields.

`redundancy` is an optional tagged-value to specify the number of runtime instances of a service. This tagged-value must be specified when a service is accessed by `Manycast` or `Anycast` filters. In Figure 10, three instances of `TransactionRecServer` are used for fault tolerance with the manycast filter `Replicator`.

Same as a connector, a service with `msgTransmissionLogRetained` and `retentionPeriod` records information on its message transmissions. A package stereotyped with `<<messageRetention>>` specifies that enclosed services have the

msgTransmissionLogRetained semantics implicitly if the services omit it (Figure 7). When a service specifies msgTransmissionLogRetained and retentionPeriod explicitly, they can overrides package's ones.

backupParameters is an optional tagged-value to specify service's backup policy. full, diff, and inc specify the time when full, differential and incremental backups is performed respectively. The class Calendar can specify a specific time in point and a repetition time. For example, when year is omitted (value zero means omission), the backup is performed every year (the date and time to perform a backup is specified by other data fields in Calendar). Full backup stores all data in a service, differential backup stores all data which have been modified since the last full backup, and incremental backup stores all data which have been modified since the last full or incremental backup. Differential backup requires much amount of storage and longer time to perform than incremental backup, but it can restore data faster. Also, data redundancy in differential backup reduces the risk of data loss. Differential and incremental backups must be used with full backup, and full backup must be performed at least once before differential or incremental backups are performed. One backup policy can have either differential or incremental backup at a time (xor). If diff and inc are omitted, only full backup is performed. generation specifies the number of full backups retained in a storage. encryptionAlgorithm specifies an algorithm to secure backup data. encryptionAlgorithm specifies an algorithm to secure backup data. securityTokens specifies security tokens for the purpose of authentication (see below). In Figure 10, the service TransactionRecServer has a backup policy. The backup policy specifies the generation (five), the time when full and incremental backup are performed (10:00pm on every Saturday and 2:00am on week days respectively).

AccessControlledService is a stereotype extending the stereotype Service (Figure 12). It is a special type of service 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 (Organization for the Advancement of Structured Information Standards, 2005). In Figure 10, TransactionRecServers control accesses from Suppliers using X.509 certificates or Kerberos tickets. 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 designer chooses. Application developers are required to complete implementing access control policies.

In addition to the general type of service, the proposed UML profile provides three special types of services, MessageConverter, MessageSplitter, MessageAggregator, to define the message processing semantics. They inherit the Service stereotype.

MessageConverter converts an incoming message with a given rule. Similar to Router, supporting tools transform a MessageConverter to a skeleton source code or rule description (e.g., in XSLT) that performs message conversions, and developers complete the skeleton code/description.

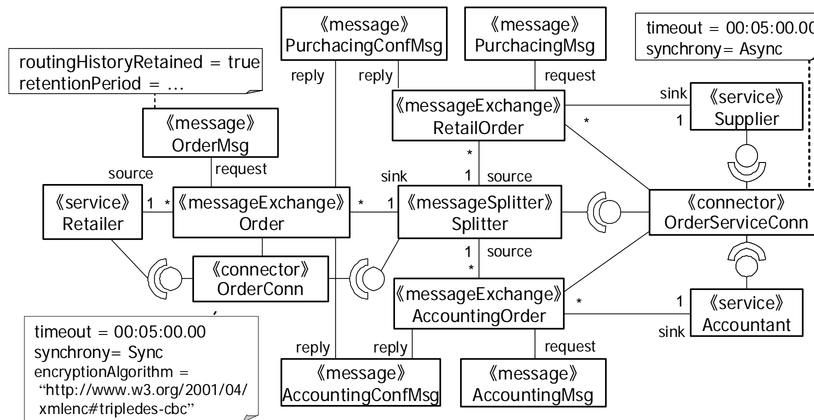

Figure 13. An Example of `MessageSplitter`

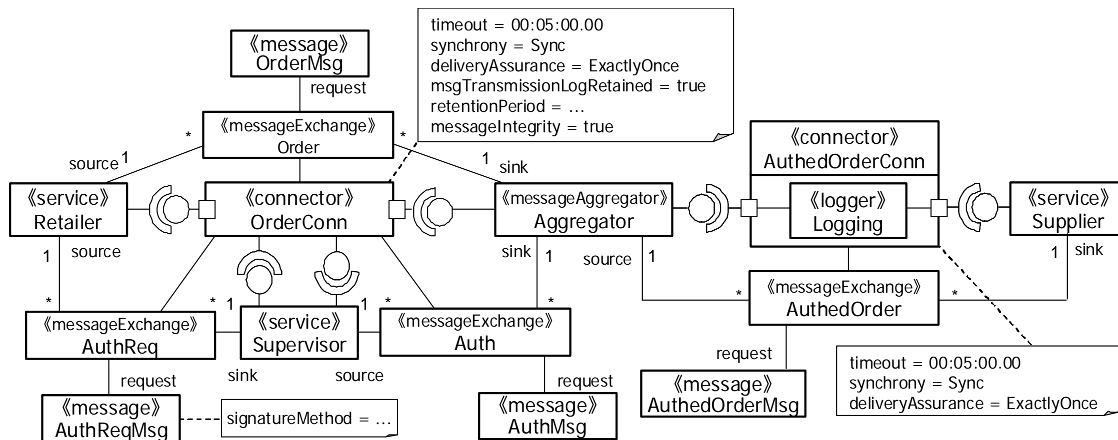

Figure 14. An Example of `MessageAggregator`

`MessageSplitter` divides an incoming message into multiple fragments with a certain rule. Similar to `MessageConverter`, supporting tools transform a `MessageSplitter` to a skeleton code or rule description that performs message split, and developers complete the skeleton. In an example model shown in Figure 13, a `Retailer` sends an order message (`OrderMsg`) to a `MessageSplitter`, and the splitter divides the message into two fragments (`PurchasingMsg` and `AccountingMsg`), and sends them to different destinations (`Supplier` and `Accountant`). The destinations directly returns reply messages (`PurchasingConf` and `AccountingConf`) to the `Retailer`. The connector `OrderConn` encrypts all messages with Triple DES. Also, the message `OrderMsg` retains routing information, which includes source of a message. (i.e., it is auditable which customer sends which message.)

`MessageAggregator` combines multiple incoming messages. Figure 14 shows an example extending the model in Figure 2. In addition to `OrderMsg`, `Retailer` sends a message `AuthReqMsg` to ask the service `Supervisor` to authorize the order. `Aggregator` synchronizes and combines `OrderMsg` and `AuthMsg` (an authorization message from `Supervisors`), and it sends new message `AuthedOrderMsg` to `Supplier`. The connector `OrderConn` retains logs on message exchanges between `Buyer`, `Supervisor` and

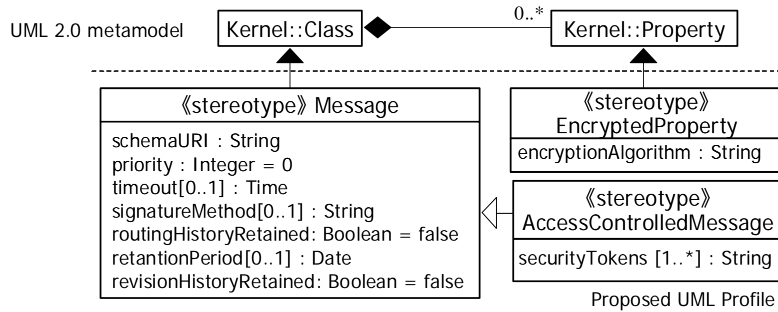

Figure 15. Tagged-Values of `Message`

`Aggregator`. It makes logs on order and authorization process auditable. Also, `OrderConn` ensure the integrity of messages.

Message

`Message` has seven tagged-values (Figure 15). `schemaURI` is a mandatory tagged-value to identify 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` are optional tagged-values to specify the priority and timeout period of messages. `Connector` has `timeout`, and `Service` also have those two tagged-values. The precedence is that `Message`'s tagged-values override `Service`'s ones, and `Service`'s tagged-values override `Connector`'s ones. Same as `Service`, each message is expected to have data fields corresponding to the `priority` and `timeout` tagged-values, and different message instance can have different `priority` and `timeout`.

`signatureMethod` is used an optional tagged-value to ensure the integrity of a message. It specifies an algorithm for generating the message's digital signature. The algorithm is represented with a URI defined in the XML Signature specification (The World Wide Web Consortium, 2002b). For example, DSA (Digital Signature Algorithm) is represented with `http://www.w3.org/2000/09/xmldsig#dsa-sha1`. In Figure 10, each `Inquiry` and `TransactionRecord` message is signed with DSA. When `signatureMethod` is specified, each message is expected to maintain its signature in a data field called `signature`.

Same as a connector, a message with `routingHistoryRetained` and `retentionPeriod` records information on its message transmissions. A package stereotyped with `<<messageRetention>>` specifies that enclosed messages have the `routingHistoryRetained` semantics implicitly if the services omit it (Figure 7). When a message specifies `routingHistoryRetained` and `retentionPeriod` explicitly, they can overrides package ones.

`revisionHistoryRetained` is an optional tagged-value to specify whether to retain message's revision history (Figure 7). It makes the revision history to auditable in the future. A message with this semantics records 1) which data fields are revised, 2) how they revised (i.e., newly created, replaced, or deleted), 3) when they revised, and 4) who revised them. The tagged-value `retentionPeriod` is used to specify the period to retain the history. A package

stereotyped with <<messageRetention>> specifies that enclosed messages have the revisionHistoryRetained semantics implicitly

The stereotype EncryptedProperty is used for message-level (end-to-end) encryption. It is defined as a stereotype extending Property in the UML metamodel (Figure 15). This stereotype is attached to data fields to be encrypted in a message. EncryptedProperty has a tagged-value, encryptionAlgorithm, to specify an algorithm used to encrypt a message. The semantics of this tagged-value is same as that of encryptionAlgorithm in Connector. An encryption algorithm is specified as a URI that the XML Encryption specification defines (World Wide Web Consortium, 2002a). Different data fields in a message can be encrypted with different encryption algorithms. In Figure 10, orders in TransactionRecordMsg are encrypted with Triple DES, which is represented with http://www.w3.org/2001/04/xmlenc#tripledes-cbc.

AccessControlledMessage is a stereotype extending Message (Figure 15). Similar to AccessControlledService it is a special type of message that enforces an access control policy. It removes the possibility of unauthorized accesses (i.e., altering messages by unauthorized users) and accidental altering (i.e. altering messages mistakenly by authorized users). The tagged-value securityTokens must be specified in AccessControlledMessage for the purpose of authentication. 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.

APPLICATION DEVELOPMENT WITH THE PROPOSED MDD FRAMEWORK

This section describes a model-driven development (MDD) tool, called Ark, which accepts a UML model designed with the proposed UML profile and transforms the model into a skeleton of application code (program code and deployment descriptor). Currently, Ark implements transformations between the proposed UML profile and three middleware technologies: Mule ESB[4], ServiceMix ESB[5] and GridFTP[6] (Figure 16). UML models in this work are maintained

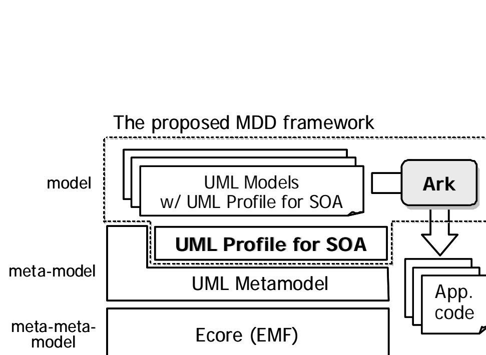

Figure 16. The Architecture of Ark

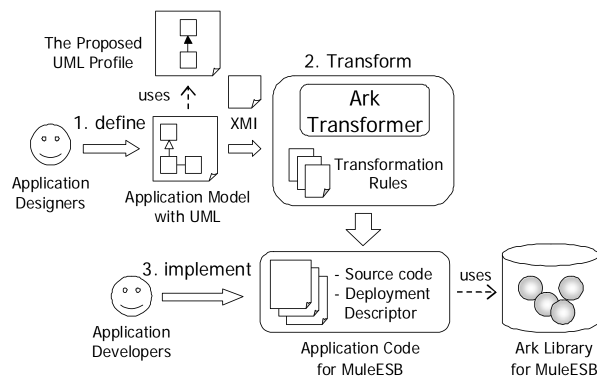

Figure 17. Application Development with Ark

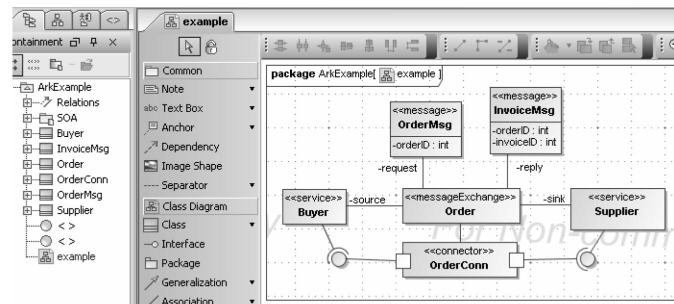

Figure 18. A UML Model in MagicDraw

with the metameta model of the Eclipse Modeling Framework (EMF; http://www.eclipse.org/emf/). The proposed UML profile is defined as an extension to the UML metamodel. Each application designer gives his/her UML model to Ark, and instructs Ark which transformation to use for generating skeleton application code.

Figure 17 shows the development process using Ark. (This figure assumes that generated application code uses Mule ESB.) Application designers define application models using the proposed UML profile (e.g., an example model in Figures 2). Ark Transformer, one of the components in Ark, takes the application models in the format of XML Metadata Interchange (XMI) and transforms the input models into application code compliant with Mule ESB.

Ark has been tested with MagicDraw[7], a visual UML modeling tool that can serialize UML models to XMI (Figure 18). Ark Transformer is implemented based on openArchitectureWare[8], a model transformation engine. Each input UML model (XMI file) is validated against the UML standard metamodel and the proposed profile's metamodel (see Figure 3), and transformed to application code for Mule ESB (Java programs and deployment descriptors in XML). A transformation rule between UML models and application code is implemented as a set of transformation templates, which define how to transform UML model elements to program elements in application code.

Transformation Rules for ESB Applications

Figure 19 shows some of the Java classes and deployment descriptors that Ark generates from the UML model in Figure 7 when Mule ESB is selected as middleware to operate applications. Table 2 shows the transformation between model elements in the proposed UML profile and program elements in Mule ESB. Ark transforms a UML class stereotyped with `<<message>>` to a Java class that has the same class name and the same data fields. The Java class implements the interface `Serializable`. This is required to implement messages exchanged with Mule ESB.

A UML class stereotyped with `<<service>>` is transformed 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 `receiveX()` to receive messages. For example, in Figure 19, `Supervisor` has `_sendOrderReqExchange()` and `receiveAuthReqExchange()` to send and receive `OrderMsg` messages to `Supplier` and from `Retailer` respectively.

[7] http://www.magicdraw.com/
[8] http://www.openarchitectureware.org/

`_sendX()` is supposed to be invoked by methods in the same service class. (This is why its visibility is private.) `receiveX()` is called by source services that have messages to deliver. A fragment of a deployment descriptor in Figure 19 specifies the URL of `Supervisor`. `<endpoint-identifier>` specifies a name of an end point (`name`) and its URL (`value`), e.g., when a service is deployed to be accessed via HTTP, `value` is `http://....` `<mule-descriptor>` specifies the implementation (`implementation`) of a service (`name`), and `<inbound-router>` specifies the URL of a service by referencing an end point.

A template fragment of this transformation rule is shown below. `<<service.name>>` in the template represents the name of a UML class stereotyped with `<<service>>`. (Note that the variables and keywords in a transformation rule are embraced with `<<` and `>>`.) `<<messageExchange.name>>` references the name of an UML class stereotyped with `<<messageExchange>>`. `<<requestMessage.name>>` and `<<replyMessage.name>>` represent the names a request and reply messages, respectively. Ark replaces each variable in a transformation template with the name of a UML model element (e.g., class name), and generates Java code. When a service transmits multiple pairs of request and reply messages, Ark generates corresponding sets of `_sendX()` and `receiveX()`. `<<sinkID>>` represents the logical name of a destination service. Each pair of a logical service name and its access point is specified in a deployment descriptor. For example, if a service is deployed to be accessed via HTTP, its access point starts with `http://`.

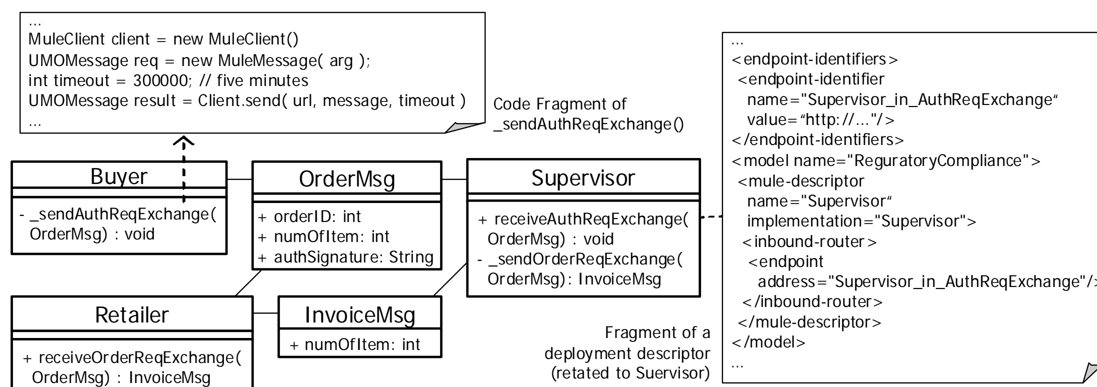

Figure 19. Generated Code for Mule ESB

```
public class <<service.name>> {
  private void _send<<messageExchange.name>>(
    <<requestMessage.name>> request){
    MuleClient muleClient = new MuleClient();
    String endpointName = <<sinkID>>;
    UMOEndpoint url =
      MuleManager.getInstance().lookupEndpoint( endpointName );
    int timeout = <<connector.timeout>>000;
    <<IF connector.synchrony == Sync>>
      UMOMessage result =
        muleClient.send( url, message, timeout );
    <<ELSEIF connector.synchrony == Oneway>>
    // generating code for an oneway call
    <<ELSE>>
```

```
    // generating code for an asynchronous call
    <<ENDIF>>
  }

  public void receive<<messageExchange.name>>(
    <<replyMessage.name>> reply){
  }
}
```

UML classes stereotyped with <<messageExchange>> and <<connector>> are not transformed to particular Java classes. The message transmission/processing semantics specified in a UML model is implemented in Java classes of message sender and destination. For example, in Figure 7, a Buyer sends an OrderMsg message to a Supervisor synchronously. Therefore, Ark generates a Java code to send the message synchronously using Mule ESB's API[9], and embeds the code in _sendAuthReqExchange() of Buyer. <<connector.synchrony>> in the above transformation template references the synchrony of a connector, and Ark interprets it to generate Java code to send messages to a destination service. (<<IF>>, <<ELSEIF>> and <<ENDIF>> are the reserved keywords for branching statements.) Ark also generates Java code to handle timeout using Mule ESB's API (<<connector.timeout>> references the timeout period specified in a UML model.), and embeds the code in _sendAuthReqExchange() of Buyer (see also Figure 19).

As Figure 7 shows, a connector has the messageIntegrity semantics. To support this semantics, Ark provides a pair of message transformers to generate and verify a message's hash value. (These transformers are implemented as a part of Ark Library. See Figure 19.) In Mule ESB, each service can have an arbitrary number of message transformers as the classes implementing the interface org.mule.transformer.UMOTransformer. Message transformers are invoked (or hooked) when a service sends/receives a message. At a message source, a transformer (edu.umb.cs.MessageIntegrityGenerator) generates a message's hash value and embeds it into the message's header. At a message sink, a transformer (edu.umb.cs.MessageIntegrityVerifier) verifies the message's integrity using the hash value. Ark Library also implements the msgTrasmissionLogRetained and routingHistoryRetained semantics as message transformers edu.cs.umb.TransmissionLogger and edu.cs.umb.RoutingHistoryLogger).

When a UML model specifies a connector as a message queue, Ark generates application code that uses Java Message Service (JMS) because Mule ESB supports message queues through the use of JMS. For example, in Figure 6, OrderConn is specified as a message queue. Ark generates a corresponding deployment descriptor to configure and establish JMS connector that exchanges OrderMsg and OrderConfirmationMsg between Retailer and InventoryManager.

When a UML model uses the MessageSplitter or MessageAggregator filter (e.g., Figures 14 and 15), Ark generates application code that uses corresponding class in Ark Library. Corresponding to MessageSplitter, Ark generates a class implementing the interface org.mule.routing.outbound.AbstractMessageSplitter. In Mule ESB, the

[9] Mule ESB provides three different APIs to send messages in synchronous, asynchronous and oneway (non-blocking) manners.

implementation class can be attached to arbitrary services in order to split an outgoing message into fragments and route them to different services. When Ark transforms a UML model in Figure 14, the implementation class is attached to a `Retailer` for intercepting an `OrderMsg` message from the `Retailer` and spliting it to a `PurchasingMsg` and `AccountingMsg`.

Similarly, corresponding to `MessageAggregator`, Ark generates a class implementing the interface `org.mule.routing.outbound.AbstractEventAggregator`. The implementation class can be attached to arbitrary services to aggregate an incoming message into a single message. In order to transform a UML model in Figure 15, Ark attaches the implementation class to a `Supplier` to aggregate a `OrderMsg` and a `AuthMsg` to a `AuthedOrderMsg` and pass the aggregated message to the `Supplier`.

The `Logger, MessageFilter, Router` and `Validator` filters are transformed to and implemented with corresponding classes built in Mule ESB. Those classes are attached to services to perform message loging, filtering, routing and validation functionalities as specified in an input UML model.

Table 2. Transformation between the Proposed UML Profile and Mule ESB

Model Element in the Proposed UML Profile	**Program Element in Mule ESB**
<<service>> <<accessControlledService>>	A Java class with the same name.
securityTokens	A security filter implemented in Ark library
<<message>>	A Java class implementing Serializable interface
signatureMethod	A security filter implemented in Ark library
<<encryptedProperty>>	A property in a corresponding Java class
encryptionAlgorithm	A message transformer implemented in Ark library
<<messageExchange>>	Methods to send/receive messages
sink (Service's role)	Service's operations sending messages.
source (Service's role)	Service's operations receiving messages.
<<connector>>	A set of entities in a deployment descriptor
timeout	An operation's parameter to specify message's timeout period
synchrony	Different types of Mule ESB's operation used to send a message.
deliveryAssurance	A filter implemented in Ark library
queueParameters	JMS parameters specified in a deployment descriptor
encryptionAlgorithm	A message transformer implemented in Ark library
msgTransmissionLogRetained	A message transformer implemented in Ark library
routingHistoryRetained	A message transformer implemented in Ark library
messageIntegrity	A message transformer implemented in Ark library
<<messageAggregator>>	A class implementing AbstractEventAggregator in Mule ESB
<<messageConverter>>	A class implementing DefaultTransformer in Mule ESB
<<messageSplitter>>	A class implementing AbstractMessageSplitter in Mule ESB
<<logger>> <<messageFilter>> <<router>> <<validator>>	Filters provided by Mule ESB
<<multicast>>	A filter implemented in Ark library
<<manycast>>	A filter implemented in Ark library
<<anycast>>	A filter implemented in Ark library

Transformation Rules for Secure and Broadband File Transfer Applications

When an application designer chooses GridFTP to operate his/her application, the application is deployed on Mule ESB and configured to use GridFTP as a message transport. Figure 20 shows some of the Java classes and deployment descriptors that Ark generates from a UML model in Figure 10.

As Figure 10 shows, the data field `records` is encrypted in `TransactionRecordMsg`. Since Mule ESB does not support message-level encryption, Ark Library provides a pair of message transformers to encrypt and decrypt data fields in messages (`edu.cs.umb.MessageEncryptor` and `edu.cs.umb.MessageDecryptor`). Ark generates a deployment descriptor to configure services so that they use those encryption/decryption transformers when they send/receive messages. Figure 20 shows a fragment of generated deployment descriptor for `TransactionRecServer`. It configures `TransactionRecServer` to use a message encryption transformer (`edu.cs.umb.MessageEncryption`) to encrypt the data field `records` in `TransactionRecordMsg` using Triple DES.

As Figure 10 shows, each `InquiryMsg` and `TransactionRecordMsg` message is signed with DSA, and `TransactionRecServer` performs authentication with X.509 or Kerberos. Since Mule ESB does not support DSA signatures and X.509/Kerberos security tokens, Ark Library provides a set of security filters to write/read signatures and security tokens by implementing the interface `org.mule.umo.security.UMOEndpointSecurityFilter`. Similar to message transformers, security filters are invoked when a service sends/receives a message. Ark generates a deployment descriptor that configures services to use the security filters Ark provides. Figure 20 shows a fragment of generated deployment descriptor for `Supplier`. It configures `Supplier` to include a DSA signature and an X.509 security token in each `Inquiry` message using two filters (`edu.cs.umb.securityfilter.Signature` and `edu.cs.umb.securityfilter.SecurityToken`).

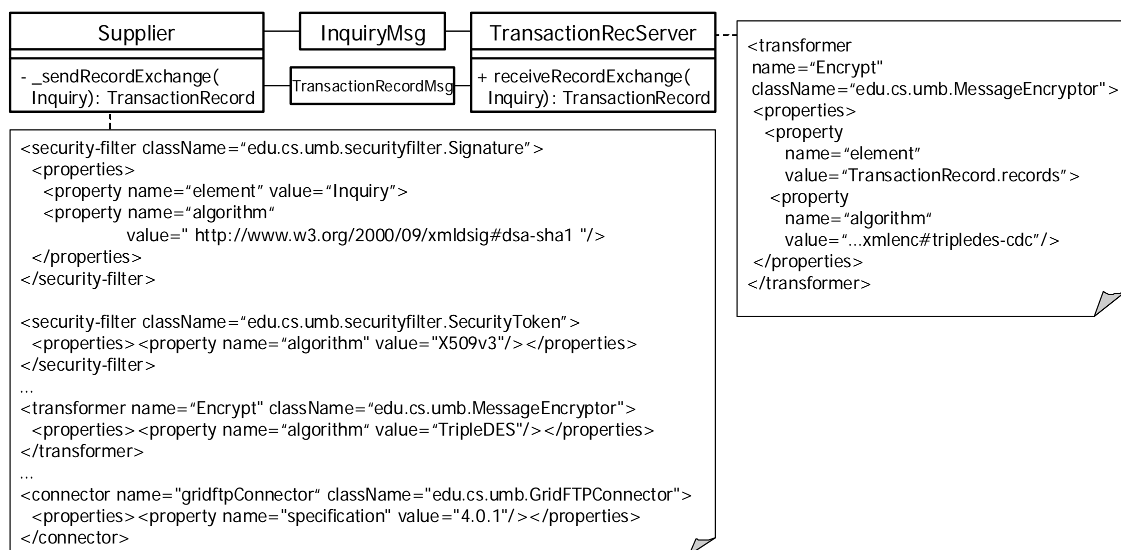

Figure 20. Generated Code for Mule ESB and GridFTP

In Figure 10, a `TransactionRecordMsg` is expected to contain a huge amount of data (e.g., scanned contract). When this example application uses GridFTP as a message transport to improve its throughput, Ark generates a deployment descriptor that configures `Supplier` and `TransactionRecordServer` to use GridFTP to transmit `InquiryMsg` and `TransactionRecordMsg` messages (Figure 20). Although Mule ESB does not support GridFTP, it provides a plug-in mechanism to implement arbitrary message transports. Ark Library implements a plug-in for GridFTP (`edu.cs.umb.GridFTPConnector`) so that services can use it in Mule ESB.

Extensibility of the Proposed MDD Framework

The proposed MDD framework (i.e., the proposed UML profile and Ark) is designed and implemented extensible. For example, application developers can change the default transformation rules that Ark provides. They can also integrate arbitrary implementation technologies with Ark in addition to currently-supposed three middleware (e.g., other ESBs and databases). These extensions can be made by changing the default set of transformation templates.

Moreover, the proposed MDD framework allows application developers to introduce arbitrary non-functional aspects that it does not support currently. Since the proposed UML profile is built on the UML standard metamodel with the standard extension mechanism (i.e., stereotypes and tagged-values), application developers can add new stereotypes and tagged-values representing their own non-functional aspects. This extension can be made by defining a set of transformation rules for new stereotypes and tagged-values. These newly-defined stereotypes/tagged-values and transformation rules have no effects on existing UML models and Ark itself (e.g., existing transformation rules, Ark Transformer and Ark Library).

Another type of extensibility of the proposed MDD framework is the ability to support arbitrary UML modeling tools. As described earlier, MagicDraw has been used as the default UML modeling tool; however, Ark can accept UML models from any modeling tools that serialize them in XMI. Choices of modeling tools have no effects on existing models and Ark.

EVALUATION

This section evaluates how the proposed MDD framework (i.e., the proposed UML profile and Ark) improves the reusability and maintainability of service-oriented applications. Given its two properties, the proposed MDD framework allows UML models (i.e., non-functional models built with the proposed profile) to be reusable across different implementation technologies. The first property is that the proposed UML profile allows application developers to model non-functional aspects in their applications in an implementation independent manner by abstracting away low-level details of implementation technologies (e.g., ESBs). As the second property, Ark can map a single UML model to different implementation technologies by switching transformation rules, even if those technologies are very different with each other. For example, Ark currently supports very different ESBs as implementation technologies: Mule ESB and ServiceMix ESB; their APIs and deployment descriptor schemata have no compatibility. The following code fragments are Java classes that Ark generates from the `Supervisor` class in Figure 7 to Mule ESB and ServiceMix ESB. In Mule ESB, a service can be implemented as a simple Java class.

```
public class Supervisor {

 public void receiveAuthReqExchange(OrderMsg reply){/*…*/}
```

```
 private void _sendOrderReqExchange (OrderMsg request){
  MuleClient muleClient = new MuleClient();
  String endpointName = …
  UMOEndpoint url = …
  Int timeout = …
  FutureMessageResult result =
    muleClient.sendAsync( url, request, timeout );
}}
```

On the other hand, in ServiceMix ESB, a service is implemented as a class that extends the `ComponentSupport` class and implements the `MessageExchangeListener` interface. Messages are received through the `onMessageExchange` method.

```
public class Supervisor extends ComponentSupport
 implements MessageExchangeListener {
 public void onMessageExchange(MessageExchange exchange)
  throws MessagingException {
  if (exchange.getRole() == Role.CONSUMER) {
   ServiceEndpoint ep = exchange.getEndpoint();
   if (ep.getServiceName().getLocalPart().equals(RETAILER)) {
    receiveAuthReqExchange(exchange);
 }}}
 private void receiveAuthReqExchange(MessageExchange exchange)
  throws MessagingException {/*…*/}
 private void _sendOrderReqExchange(OrderMsg orderMsg){
  InOut inout = createInOutExchange(SUPPLIER, null, null);
  NormalizedMessage msg = inout.createMessage();
  // ...
  inout.setInMessage(msg);
  sendSync(inout);
 }}
```

By making UML models (i.e., non-functional models) reusable across different implementation technologies, the proposed MDD framework allows application developers to reuse or repurpose services without knowing the details of implementation technologies.

Table 3 shows the program elements (Java code and/or deployment descriptors: DD) that Ark generates for Mule ESB and ServiceMix ESB from a single UML model element. Table 3 also shows the lines of code (LOC) of each generated program element. (LOC is shown in parentheses.) As this figure illustrates, a single model element represents multiple program elements in the proposed MDD framework. For example, `queueParamters` represents 34 LOC in Mule ESB and 33 LOC in ServiceMix ESB. This contributes to improve the maintainability of service-oriented applications by freeing application developers from manually and carefully dealing with many lower-level program elements in a consistent manner.

RELATED WORK

This paper is a set of extensions to the authors' prior work (Wada et al., 2006a; Wada et al., 2006b; Wada et al., 2006c). As one of the extensions, this work investigates new non-functional aspects for regulatory compliance, which were beyond of the scope of the prior work. Another

Table 3. Generated Program Elements and their LOC

Model Elements in the Proposed UML Profile	Program Elements and their LOC in Mule ESB	Program Elements and their LOC in ServiceMix ESB
<<service>> <<accessControlledService>>	A Java class (8) An endpoint identifier in DD (1) A service entry in DD (7)	A Java class (9) A service entry in DD (6)
securityTokens	An in-bound filter in DD (3)	An in-bound filter in DD (3)
<<message>>	A Java class (2)	A Java class (2)
signatureMethod	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (6)
<<encryptedProperty>>	An attribute in a Java class (1)	An attribute in a Java class (1)
encryptionAlgorithm	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (10)
<<messageExchange>>	In-bound and out-bound routers in DD (6)	A routing conf. in DD (14)
sink (Service's role)	A method to send in Java (10)	A method to send in Java (10)
source (Service's role)	A method to receive in Java (2)	A method to receive in Java (12)
<<connector>>	No code generated (0)	No code generated (0)
synchrony	Java code in Mule ESB API (1)	Java code in ServiceMIX API (1)
deliveryAssurance	A configuration entry in DD (3)	A configuration entry in DD (6)
queueParameters	A configuration entry in DD (14) A JMS configuration file (20)	A configuration entry in DD (6) JNDI configuration in DD (7) A JMS configuration File (20)
encryptionAlgorithm	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (10)
msgTransmissionLogRetained	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (6)
routingHistoryRetained	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (6)
messageIntegrity	In-bound and out-bound filters in DD (6)	In-bound and out-bound filters in DD (6)
<<messageAggregator>>	A Java class (4) An In-bound filter in DD (3)	A Java class (4) An endpoint conf. in DD (2)
<<messageConverter>>	A Java class (4) An out-bound filter in DD (3)	A Java class (4) An endpoint conf. in DD (2)
<<messageSplitter>>	A Java class (4) An out-bound filter in DD (3)	A Java class (4) An endpoint conf. in DD (2)
<<logger>>	An out-bound filter in DD (3)	An out-bound filter in DD (3)
<<messageFilter>>	An out-bound filter in DD (3)	A filter conf. in DD (7)
<<router>>	An out-bound filter in DD (3)	A routing conf. in DD (7)
<<validator>>	An out-bound filter in DD (3)	An out-bound filter in DD (3)
<<multicast>>	An out-bound filter in DD (3)	A routing conf. in DD (7)
<<manycast>>	An out-bound filter in DD (3)	A routing conf. in DD (7)
<<anycast>>	An out-bound filter in DD (3)	A routing conf. in DD (7)

extension is that Ark currently supports a wider range of implementation technologies. As a result, the proposed MDD framework now allows application developers to model SOA’s non-functional aspects through hiding the implementation differences across two of the most major ESBs (Mule ESB and ServiceMix ESB). Given these extensions, this paper fully discusses the updated details in the design and implementation of the proposed MDD framework. Moreover,

unlike the prior work, this work empirically evaluates how the proposed MDD framework contributes to the reusability and maintainability of service-oriented applications.

There are several UML profiles proposed for SOA. Marcos et al. (2003) and Amsden et al. (2005) propose UML profiles to specify functional aspects in SOA. Both profiles are designed based on the XML schema of Web Service Description Language (WSDL). Each profile provides a set of stereotypes and tagged-values that correspond to the elements in WSDL, such as `Service`, `Port`, `Messages` and `Binding`[10]. Since WSDL is designed to define only functional aspects of web services, non-functional aspects are beyond of the scope of Marcos et al. (2003) and Amsden et al. (2005). Ermagan et al. (2007) proposes and Object Management Group (2006b) standardizes UML profiles for functional aspects in SOA. Unlike the above profiles, the proposed profile focuses on specifying non-functional aspects in SOA.

Amir et al. (2005) propose a UML profile to describe both functional and non-functional aspects in SOA. This profile is generic enough to specify a wide range of non-functional aspects. For example, the stereotypes for non-functional aspects include `<<policy>>` and `<<permission>>`. However, their semantics tend to be ambiguous. This profile does not precisely define what non-functional aspects developers can (or are supposed to) specify and how to represent them with tagged values in accordance with given stereotypes. Ortiz et al. (2006) also propose a generic UML profile to describe various non-functional aspects (called extra-functional properties). Arbitrary non-functional aspects can be defined as stereotypes extending the `<<Extra-Functional Property>>` stereotype. However, it is ambiguous how to define particular non-functional aspects with user-defined stereotypes and tagged-values. The World Wide Web Consortium (2006) standardizes the WS-Policy specification, a generic XML format to specify arbitrary non-functional aspects of web services. No explicit principles and guidelines are available on how to define particular non-functional aspects with XML document elements. Unlike the above three schemes, the proposed UML profile carefully and precisely defines a variety of stereotypes and tagged-values for non-functional aspects in SOA so that the proposed MDD tool (Ark) can interpret and transform models to code in an unambiguous manner.

Vokäc (2005) proposes a UML profile for data integration in SOA. It provides data structures to specify messages. Application developers can use the data structures for building dictionaries that maintain message data used in existing systems and new applications. This profile separates data integration as a non-functional aspect from functional aspects, and enables specifying data integration in an implementation independent manner. This UML profile and the proposed profile focus on different issues in SOA. Data integration is beyond of the scope of the proposed profile, and Vokäc (2005) does not consider non-functional aspects in message transmission, message processing, security and service deployment.

Heckel et al. (2003) propose a UML profile for dynamic service discovery in SOA. This profile provides a set of stereotypes (e.g., `<<uses>>`, `<<requires>>` and `<<satisfies>>`) to specify relationships among service interfaces, service implementations and functional requirements. For examples, a relationship can specify that a service *uses* other services, and another relationship can specify that a service *requires* other services that *satisfy* certain functional requirements. These relationships are intended to aid dynamic discovery of services.

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

Rather than service discovery, the proposed UML profile focuses on non-functional semantics in message transmission, message processing, security and service deployment.

Object Management Group (2007) standardizes a UML profile for Data Distribution Service (DDS). DDS is a standard specification for publish/subscribe middleware, and it supports several non-functional aspects in real-time messaging. OMG's UML profile for DDS allows UML models to specify these non-functional aspects. In contrast, the proposed profile is not limited to real-time messaging, but supports a wider range of non-functional aspects. Moreover, OMG's profile is designed to be mapped into only DDS implementations. In contrast, the proposed profile is designed in an implementation independent manner; it can be mapped to arbitrary implementation technologies.

Gardner (2003), List et al. (2005), Johnston (2004) and Object Management Group (2005a) define UML profiles to specify service orchestration and map it to Business Process Execution Language (BPEL) (Organization for the Advancement of Structured Information Standards, 2003). 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.

Lodderstedt et al. (2002) propose a UML profile, called SecureUML, to define role-based access control for network applications. SecureUML provides stereotypes to assign roles (<<security.role>>) and access control permissions (<<security.constraint>>) to applications (e.g., UML interfaces and classes). SecureUML uses Object Constraint Language (OCL) to define access control. Jürjens (2002) propose another UML profile, called UMLsec, to define data encryption (<<data security>>) and secure network links (<<encrypted>>). Wang et al. (2005) and Nakamura et al. (2005) also propose UML profiles to define security aspects. In addition to security aspects, Soler et al. (2006) propose a UML profile extending the Common Warehouse Metamodel (Object Management Group, 2003) in order to define regulatory audit policies in data warehouses. For example, the profile provides stereotypes to specify whether a data warehouse retains logs to access data sources. Gönczy et al. (2006) propose a formal definition of reliable messaging mechanisms as a metamodel. These profiles/metamodels are parallel to the proposed UML profile in terms of the ability to describe security aspects, audit policies and reliable messaging in network applications. However, the proposed UML profile covers not only security, auditing or reliable messaging aspects but also many other non-functional aspects in SOA (e.g., message queuing, message validation/filtering, and message.

Zhu et al. (2007) and Zou et al. (2006) propose UML profiles to visually define non-functional requirements such as desirable response time and throughput. However, they do not consider model transformation to map non-functional requirements to certain implementation technologies. In contrast, the proposed profile is designed to consider model transformation, although non-functional requirements are beyond of the scope of the proposed profile.

There are several specifications and research efforts to investigate implementation techniques for non-functional aspects in SOA (Organization for the Advancement of Structured Information Standards, 2003; Organization for the Advancement of Structured Information Standards, 2004a; Organization for the Advancement of Structured Information Standards, 2004b; Baligand et al., 2004; Wang et al., 2004; Mukhi et al., 2004). 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 investigating specific implementations of non-functional aspects in SOA, the proposed MDD framework is intended to

provide a means for application developers to model and maintain non-functional aspects in an implementation independent manner so that they can be mapped on different specifications or implementation technologies.

CONCLUDING REMARKS

This paper proposes a model-driven development (MDD) framework for non-functional aspects in SOA. The proposed MDD framework consists of (1) a UML profile to graphically specify and maintain SOA non-functional aspects in an implementation independent manner, and (2) an MDD tool that accepts a UML model defined with the proposed profile and transforms it to application code (program code and deployment descriptors). This paper presents design details of the proposed UML profile, and describes how the proposed MDD tool uses the profile to develop service-oriented applications that can run with different implementation technologies such as Mule ESB, ServiceMix and GridFTP. Empirical evaluation results show that the proposed MDD framework contributes to improve the reusability and maintainability of service-oriented applications by hiding the details of implementation technologies.

Several extensions to the proposed MDD framework are planned as future work. As described in the Related Work section, there are several other UML profiles for SOA. The proposed profile will be co-used or integrated with some of them (e.g., Oba et al., 2005; Object Management Group, 2005b) in order to investigate a more comprehensive development framework for SOA.

Another extension is to integrate the proposed UML profile with a modeling language for business processes such as Business Process Modeling Notation (Object Management Group, 2006a). The proposed profile is designed to specify applications from a structural point of view; it does not consider a viewpoint of processes or workflows. Therefore, as the size of a model (application) increases, it becomes harder to understand how messages are exchanged among services and define non-functional aspects along with message flows. For example, in order to specify secure messaging for a certain business process (e.g., order processing process), it can be time-consuming and error-prone to find all the services associated with the process and define a security aspect for the connections among those services. The integration with a business process modeling language can make non-functional modeling more intuitive by providing both structural and process viewpoints.

The proposed MDD framework will be evaluated in several different application domains[11]. One of them is service-oriented system integration in a natural gas utility company. The proposed UML profile and Ark are planned to be used in a system integration project, and their design and implementation will be enhanced through the project experience. Another application domain is eco-informatics. The proposed framework has been used to design and maintain ecological observation systems (Wada et al., 2006d). Ecological observation systems monitor ecosystems, record various observation data (e.g., a niche of a particular species and weather in the niche), and help ecologists understand and predict the observation of ecosystems. Currently, ecological observation systems are often implemented monolithic; their extensibility and customizability are limited. SOA is expected to overcome this issue by decomposing an observation system into multiple services, implementing the system as a combination of services, and extending/customizing it through a recombination of services (Bermudez et al., 2006). The proposed MDD framework has been used to separate functional and non-functional aspects in an

[11] A software engineering discipline suggests investigating at least three applications on a framework in order to examine the framework’s generality and reusability. (Roberts et al., 1997)

ecological observation system and model/implement non-functional aspects in the system. Through this practice, the proposed MDD framework will be enhanced to improve its generality.

ACKNOWLEGMENT

This work is supported in part by OGIS International, Inc.

REFERENCES

Amir, R., & Zeid, A. (2004). A UML profile for service oriented architectures. ACM Object-Oriented Programming, Systems, Languages, and Applications Poster session.

Amsden, J., Gardner, T., Griffin C., & Iyengar, S. . (2005). UML 2.0 profile for software services. IBM developerWorks.

Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., & Foster, I. (2005). The Globus striped GridFTP framework and server. ACM Super Computing.

Arsanjani A., Zhang L., Ellis M., Allam A., & Channabasavaiah K. (2007) S3: A service-oriented reference architecture. IT Professional, 9(3).

Baligand, F., & Monfort, V. (2004). A concrete solution for web services adaptability using policies and aspects. UNITN/Springer International Conference on Service Oriented Computing.

Bermudez, L., Bogden, P., Bridger, E., Creager, G., Forrest, D., & Graybeal, J. (2006). Toward an ocean observing system of systems. MTS/IEEE Oceans.

Bichler, M., & Lin, K. (2006). Service-oriented computing. IEEE Computer, 39(6).

Bieberstein N., Bose S., Fiammante M., Jones K., & Shah R. (2005). Service-oriented architecture (SOA) compass: business value, planning, and enterprise roadmap. IBM Press.

Chappell, D. (2004). Enterprise service bus. O'Reilly.

Ermagan, V., & Krüger, H. (2007). A UML2 profile for service modeling. ACM/IEEE International Conference on Model Driven Engineering Languages and Systems.

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., & Newling, T. (2004). Patterns: service-oriented architecture and web services. IBM Red Books.

Foster, I. (2005). Service-oriented computing. Science, 308(5723).

Fuentes, L., & Vallecillo, A. (2004). An introduction to UML profiles. The European Journal for the Informatics Professional, 5(2).

Gardner, T. (2003). UML modeling of automated business processes with a mapping to BPEL4WS. European Conference on Object-Oriented Programming Workshop on Object Orientation and Web Services.

Gönczy, L., & Varró, D. (2006). Modeling of reliable messaging in service oriented architecture. Telcert International Workshop on Web Services Modeling and Testing.

Heckel, R., Lohmann, M., & Thöne, S. (2003). Towards a UML profile for service-oriented architectures. Workshop on Model Driven Architecture: Foundations and Applications.

Java Community Process. (2001). UML profile for Enterprise Java Beans.

Johnston, S.. (2004). UML 1.4 profile for software services with a mapping to BPEL 1.0. IBM developerWorks.

Jürjens, J. (2002). UMLsec: extending UML for secure systems development. ACM/IEEE International Conference on Unified Modeling Language

Lewis, G., Morris, E., Brien, L., Smith, D., & Wrage, L. (2005). Smart: the service-oriented migration and reuse technique. Technical report, Software Engineering Institute, Carnegie Mellon University.

List, B., & Korherr, B. (2005). A UML 2 profile for business process modelling. ACM International Conference on Conceptual Modeling Workshop on Best Practices of UML at the International Conference on Conceptual Modeling.

Lodderstedt, T., Basin, D., & Doser, J. (2002). SecureUML: a UML-based modeling language for model-driven security. ACM/IEEE International Conference on Unified Modeling Language.

Marcos, E., Castro, V., & Vela, B. (2003). Representing web services with UML: a case study. UNITN/Springer International Conference on Service Oriented Computing.

Mukhi, N., Konuru, R., & Curbera, F. (2004). Cooperative middleware specialization for service oriented architectures. ACM International World Wide Web Conference.

Nakamura, Y., Tatsubori, M., Imamura, T., & Ono, K. (2005). Model-driven security based on a web services security architecture. IEEE International Conference on Services Computing.

Organization for the Advancement of Structured Information Standards. (2003). Web services business process execution language.

Organization for the Advancement of Structured Information Standards. (2004a). Web service reliability 1.1.

Organization for the Advancement of Structured Information Standards. (2004b). Web service reliable messaging.

Organization for the Advancement of Structured Information Standards. (2005). Web services security policy language.

Oba, K., Hashimoto, M., Fujikura, S., & Munehira, T. (2005). The status quo and challenges of service-oriented architecture based application design. IPSJ Workshop on Software Engineering.

Object Management Group. (2003). Common warehouse metamodel, version 1.1.

Object Management Group. (2004). UML 2.0 super structure specification.

Object Management Group. (2005a). Business process definition metamodel.

Object Management Group. (2005b). UML profile for modeling quality of service and fault tolerance characteristics and mechanisms.

Object Management Group. (2006a). Business process modeling notation, version 1.0.

Object Management Group. (2006b). UML profile and metamodel for services, request for proposal.

Object Management Group. (2006c) UML profile for data distribution service, request for proposal.

Object Management Group. (2007) Data Distribution Service for Real-time Systems, version 1.2.

O'Grady, S. (2004). SOA meets compliance: compliance oriented architecture. White paper, RedMonk.

Ortiz, G., & Hernández, J. (2006). Toward UML profiles for web services and their extra-functional properties. IEEE International Conference on Web Services.

Papazoglou, M. (2003). Service-oriented computing: concepts, characteristics and directions. IEEE International Conference on Web Information Systems Engineering.

Roberts, D., & Johnson, R. (1997). Evolving frameworks: a pattern language for developing object-oriented frameworks. Pattern Languages of Program Design 3, Chapter 26. Addison Wesley.

Soler, E., Villarroel, R., Trujillo, J., Medina, E., & Piattini, M. (2006). Representing security and audit rules for data warehouses at the logical level by using the common warehouse metamodel.IEEE International Conference on Availability, Reliability and Security.

Vinoski, S. (2003). Integration with web services. IEEE Internet Computing, 7(6).

Vokäc, M. (2005). Using a domain-specific language and custom tools to model a multi-tier service-oriented application - experiences and challenges. ACM/IEEE International Conference on Model Driven Engineering Languages and Systems.

Wang, L., & Lee, L. (2005). UML-based modeling of web services security. IEEE European Conference on Web Services, poster paper.

Wang, G., Chen, A., Wang, C., Fung, C., & Uczekaj, S. (2004). Integrated quality of service (QoS) management in service-oriented enterprise architectures. IEEE Enterprise Distributed Object Computing Conference.

Wada, H., Suzuki, J., & Oba K. (2006a). A model-driven development framework for non-functional aspects in service oriented grids. IEEE International Conference on Autonomic and Autonomous Systems.

Wada, H., Suzuki, J., & Oba K. (2006b). Modeling non-functional aspects in service oriented architecture. IEEE International Conference on Services Computing.

Wada, H., Suzuki, J., & Oba K. (2006c). A service-oriented design framework for secure network applications. IEEE International Conference on Computer Software and Applications Conference

Wada, H. & Suzuki, J. (2006d). Designing Ecological Observation Systems using Service Oriented Architecture. Elsevier/ISEI International Conference on Ecological Informatics, poster paper.

World Wide Web Consortium. (2002a). XML encryption syntax and processing.

World Wide Web Consortium. (2002b). XML signature syntax and processing.

World Wide Web Consortium. (2006). Web services policy framework.

Zhang, Z., & Yang, H. (2004). Incubating services in legacy systems for architectural migration. IPSJ/IEEE Asia-Pacific Software Engineering Conference.

Zhu, L., & Gorton, I. (2007) UML profiles for design decisions and non-functional requirements. ACM/IEEE International Conference on Software Engineering Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent.
Zou, J., & Pavlovski, C. (2006) Modeling architectural non functional requirements: from use case to control case. IEEE International Conference on e-Business Engineering.

ABOUT THE AUTHORS

Hiroshi Wada Hiroshi Wada received his M.S. degree in computer science from Keio University, Japan, in 2002. He started the Ph.D. program in the University of Massachusetts, Boston in 2005. His research interests include model-driven software development, and service oriented architecture. Before enrolling in the Ph.D. program, he worked for Object Technology Institute, Inc., and engaged in consulting and educational services in the field of object oriented technologies, distributed systems and software modeling.

Junichi Suzuki Junichi Suzuki received a Ph.D. in computer science from Keio University, Japan, in 2001. He joined the University of Massachusetts, Boston in 2004, where he is currently an Assistant Professor of computer science. From 2001 to 2004, he was with the School of Information and Computer Science, the University of California, Irvine (UCI), as a postdoctoral research fellow. Before joining UCI, he was with Object Management Group Japan, Inc., as Technical Director. His research interests include model-driven software development, autonomous adaptive distributed systems, biologically-inspired software adaptation and self-organizing sensor networks. He is an editor of International Journal of Software Patterns, International Journal of Software Architecture and International Journal of Software Reuse. He was the Program Co-Chair of the 2008 IEEE International Workshop on Methodologies for Non-functional Properties in Services Computing. He served on over 35 conference program committees, and served as a workshop co-chair for the ACM/IEEE/ICST/Create-Net BIONETICS 2007 conference. He is an active participant and contributor in the ISO SC7/WG19 and the Object Management Group, Super Distributed Objects SIG. He is a member of IEEE and ACM.

Katsuya Oba Katsuya Oba received the Batchelor of Arts degree from Osaka University, Osaka, Japan, in 1989. He joined Osaka Gas Information System Research Institute Co., Ltd. (OGIS-RI) as a systems engineer. From 2000 to 2005, he worked for OGIS International, Inc. in Palo Alto, California as General Manager and leaded several software product development and R&D projects. He returned to OGIS-RI in 2006, and is leading R&D and business development relating to Service Oriented Architecture (SOA). His research interests include software architecture, business and systems modeling and software development processes. He is a member of Information Processing Society of Japan.