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

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

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

Abstract

Service-oriented grids are grid computing systems built with the notion of service-oriented architecture (SOA). Using two major abstract concepts, services and connections between services, each grid application is designed as a collection of loosely-coupled services in an implementation independent manner. In service-oriented grids, the non-functional aspects of services and connections (e.g., message transmission and security) should be described separately from their functional aspects because different applications use services and connections in different non-functional requirements. This paper presents a UML profile to graphically specify and maintain non-functional aspects in service-oriented grids as implementation independent UML models. The UML models are transformed into implementation specific code by a model-driven development tool. This paper demonstrates how the proposed UML profile is used in model-driven development of service-oriented grid applications.

1. Introduction

Service-oriented grids are grid computing systems built with the notion of service-oriented architecture (SOA) [7, 10]. Similar to traditional grids, service-oriented grids share and virtualize distributed resources (e.g., CPUs), and coordinate them for applications on demand. In addition, unlike traditional grids, service-oriented grids aim to decompose a monolithic application to a set of loosely-coupled services. This allows application designers to reuse and integrate services on demand for building their applications in a cost effective manner. SOA is a driving force for this thrust.

In order to make on-demand service integration a reality, SOA is intended to improve the reusability and maintainability of services [5, 6, 16]. It is an architectural style to design applications in an implementation independent manner using two major abstract concepts: *services* and *connections between services*. Each service represents a functionality in an application, or encapsulates the function of a subsystem in an existing system. Each connection defines how services are connected with each other and how messages (or data) are exchanged through the connection. SOA hides the implementation details of services and connections (e.g., programming language and remoting middleware) from application designers. They can reuse and combine services to build their applications without knowing the implementation details of services and connections.

In SOA, the non-functional aspects of services and connections (e.g., message transmission and security) should be defined separately from their functional aspects because different applications use services and connections in different non-functional contexts. For example, an application may use a service via reliable connection that guarantees message delivery when the service is hosted on an unreliable network (e.g., the Internet). Another application may use the service via connection that does not guarantee message delivery when the service is reliably hosted in house. The separation of functional and non-functional aspects improves the reusability of services and connections. It also improves the ease of understanding application design and enables two different aspects to evolve independently. This results in higher maintainability of applications.

Non-functional aspects should also be captured as abstract models in an early development phase and automatically transformed to code or configuration files in order to improve development productivity [22, 23]. It incurs time-consuming and error-prone manual efforts to implement and deploy non-functional aspects in later development phases (e.g., integration and test phases) [22, 23].

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 UML profile and transforms it to application code (program code and deployment descriptors). The proposed UML profile allows application designers 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 using certain implementation technologies such as Enterprise Service Buses [4] and GridFTP [1].

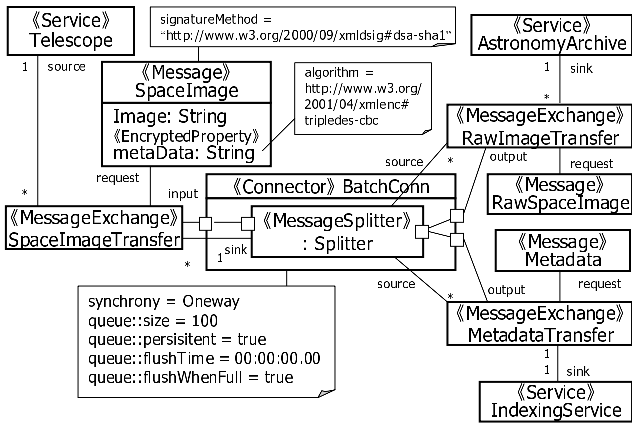
This paper is structured as follows. Section 2 motivates the proposed UML profile with an example. Section 3 describes the design details of the proposed profile. Section 4 presents how Ark is used in model-driven development of service-oriented grid applications. Sections 5 and 6 conclude with discussion on related work.

2. Background and a Motivating Example

UML provides extension mechanisms (e.g., stereotypes and tagged-values) to specialize the standard model elements (e.g., class and association) to precisely describe domain or application specific concepts [21, 8]. A stereotype is applied to a standard model element, and specializes its semantics to a particular domain or application. Each stereotyped model element can have data fields, called tagged-values, specific to the stereotype. Each tagged-value consists of a name and value. A particular set of stereotypes and tagged-values is called a UML profile.

Figure 1 shows an example model defined with the proposed UML profile. It illustrates an astrophysical virtual observatory system in which telescopes store space images in distributed archives and an indexing service maintains the images' metadata (e.g., date, latitude and longitude). In this example, three services (`Telescope`, `AstronomyArchive` and `IndexingService`) exchange messages. Each service is defined as a class decorated with the stereotype ≪`Service`≫. Services exchange three types of messages (`SpaceImage`, `RawSpaceImage` and `Metadata`), each of which is stereotyped with ≪`Message`≫. The data field steteotyped with ≪`EncryptedProperty`≫ in `SpaceImage` is encrypted with the algorithms specified as tagged-values (`algorithm = ...`).

Each pair of a request and reply messages is represented by ≪`MessageExchange`≫. ≪`Connector`≫ represents a connection that transmits messages between services. In this example, messages are delivered through the connector `BatchConn`. Every message exchange is bound with a connector in order to specify which connector is used to deliver messages. Figure 1 shows that a `Telescope` sends a `SpaceImage` message through `BatchConn`. In `BatchConn`, a `SpaceImage` is split into two messages, `RawSpaceImage` and `Metadata`, and they are delivered to an `AstronomyArchive` and `IndexingService`, respectively. In addition, `BatchConn` works as a queue that stores



《Service》 Telescope
signatureMethod = "http://www.w3.org/2000/09/xmldsig#dsa-sha1"
《Service》 AstronomyArchive
1
source
《Message》 SpaceImage
Image: String
《EncryptedProperty》
metaData: String
algorithm = http://www.w3.org/2001/04/xmlenc#tripledes-cbc
1
sink
*
《MessageExchange》 RawImageTransfer
source
*
output
request
request
input
《Connector》 BatchConn
《Message》 RawSpaceImage
*
《MessageExchange》 SpaceImageTransfer
《MessageSplitter》 : Splitter
《Message》 Metadata
*
1
sink
source
*
output
request
《MessageExchange》 MetadataTransfer
synchrony = Oneway
queue::size = 100
queue::persisitent = true
queue::flushTime = 00:00:00.00
queue::flushWhenFull = true
1
1
sink
《Service》 IndexingService

Figure 1. An Example UML Model

and forwards `SpaceImage` messages in a batch mode.

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

3. Design of the Proposed UML Profile

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

Figure 2 shows how the proposed UML profile defines its stereotypes by extending the UML metamodel. Each stereotype is defined as a metaclass stereotyped with ≪`stereotype`≫. Except `Connector`, four stereotypes inherit the `Class` metaclass in the `Kernel` package of the UML metamodel. Thus, they are applied to classes in user-defined models (see Figure 1). `Service` and `Filter` 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`/`Filters` (see Figures 1 and 2). `MessageExchange` can indicate one-to-many, many-to-one and many-to-many message exchanges, using multiplicity on two associations between a `MessageExchange` and `Services`/`Filters`. For example, Figures 1 shows an example of many-to-one message exchange between `Telescopes` and a `IndexingService`.

`Connector` is a stereotype extending the `Class` metaclass in the `InternalStructures` package of the UML metamodel (Figure 2). This metaclass defines a composite class, a special type of class, which can contain other model elements (e.g., inter classes)[1] and have `Ports` to indicate how internal model elements interact with external elements. In the proposed profile, a `Connector` can

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

contain multiple 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.

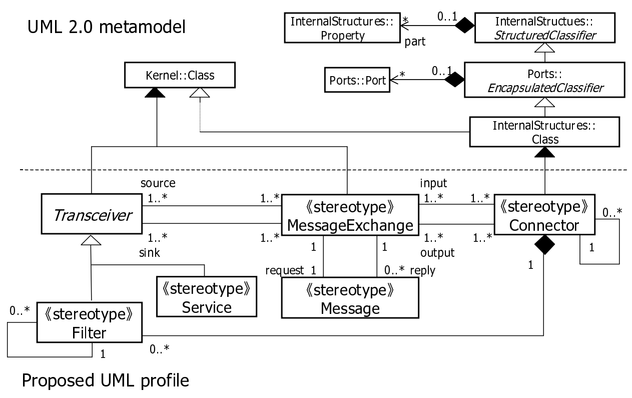


Figure 2. Definition of Stereotypes

3.1. Connector

Connector has six tagged-values (Figure 3). timeout is a mandatory tagged-value to specify the timeout period (in millisecond) in which a connector needs to deliver each message. If a message is not delivered to its destination (sink) within the timeout period, a connector discards the message.

synchrony is a mandatory tagged-value to specify the synchrony semantics of message transmissions between a message source and destination. Synchronous, asynchronous and oneway non-blocking semantics are defined as an enumeration in Synchrony (Figure 3), and each connector chooses one of them. In Figure 1, the BatchConn connector implements a oneway non-blocking semantics for message transmissions. Telescopes are not blocked when they send messages.

inOrder ismandatory tagged-value to specify whether the order of messages that a service (destination) receives is same as the order of messages that a service (source) sends.

deliveryAssurance is an optional tagged-value to specify the assurance level of message delivery. Three different semantics are defined as an enumeration in DeliveryAssurance (Figure 3), and each connector chooses one of them at a time. AtLeastOnce means that a connector retries sending a message until its destination receives the message. However, the message may be duplicated and delivered to its destination more than once. This option ignores timeout tagged-value. AtMostOnce means that a connector discards a message if it has been sent to its destination, but there is no guarantee of message delivery. ExactlyOnce satisfies the requirements of the above both options. It guarantees that a connector delivers a message to its destination without duplication.

encryptionAlgorithm is an optional tagged-value used for transport-level encryption in a connector. (see Section 3.4 for message-level encryption) This tagged-value defines an algorithm to secure a connection upon which request and response messages are transmitted. The encryption algorithm is specified as a URI defined in the XML Encryption specification [24].

queueParameters is an optional tagged-value to deploy a message queue between services (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 the value of flushWhenFull is false, the queue discards a message according to discardPolicy; 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 discardPolicy (Figure 3). 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. perisistent specifies whether a queue stores messages in a storage (e.g., in a file or database) so that the queue can recover them after it crashes unexpectedly. When the value of subscriptionRecovery is true, a queue stores all messages that cannot be delivered to a destination and transmits them when the destination reconnects to the queue.

An example model in Figure 1 shows that a connector (BatchConn) works as a queue that stores and forwards SpaceImage messages in a batch mode between Telescopes and an AstronomyArchive. The queue's length is 100 and stored in a persistent storage. The queue flushes messages to an AstronomyArchive at 0:00am everyday or when it overflows.

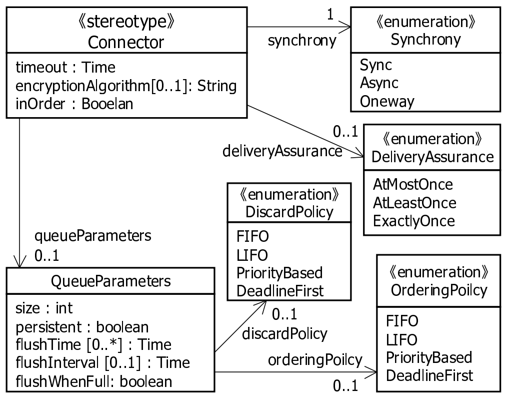


Figure 3. Tagged-Values of Connector

3.2. Filter

This paper describes six of the filters that the proposed UML profile defines. Filters are defined as stereotypes extending the `Filter` stereotype (Figure 4). New filters can be defined as its subclasses. This section shows four filters to specify message transmission semantics and a filter to specify message processing semantics.

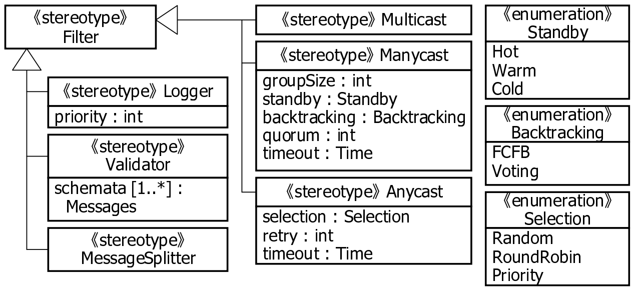


Figure 4. Tagged-Values of Filters

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

`Manycast` is used for fault tolerance of services by forwarding a request message to a group of replicated destinations (i.e., to the same type of services). `groupSize` specifies how many services are deployed as a group (Figure 4). `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 request message, out of multiple replies from services. `backtracking` defines two policies to decide which reply message to be returned (Figure 4). 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, a `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 a group and sends a message to the service. In cold standby, `backtracking` is not used.

`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. The destination is selected randomly, on round robin or destination's priority basis (the service with the highest priority in a group is selected). `retry` specifies the maximum number of retries to forward a request message. If `Anycast` fails to deliver a request message within `timeout`, it returns an error message to its source.

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

In addition to the filters regarding message transmission semantics, the proposed UML profile provides several other filters to specify message processing semantics in a connector. This paper describes two of them: `Validator` and `MessageSplitter` (Figure 4). `Validator` validates an incoming message against the schemata specified in `schemata`, and transmits only validated messages. When a connector is encrypted with `encryptionAlgorithm`, a `Validator` in the connector cannot validate messages (all messages are transmitted to their destinations).

Figure 5 shows an astrophysical virtual observatory system as well as Figure 1. In this example, through an user interface (`Portal`), an user submits a request (`AnalyseReq`) to an analyze service (`SpectralAnalyser`) to analyse space images. The analyse service accesses the image indexing serivce (`IndexingService`) to acquire relevant space images (`RawSpaceImage`), and sends back a result (`VisualImage`) to the user. In Figure 5, the connector `AnalyseConn` uses a validator to validate all messages, and also the connector `QueryConn` uses a logger to record all `ImageQuery` messages.

`MessageSplitter` divides an incoming message into multiple fragments with a certain rule. Since UML does not provide a good means to define rules, the proposed UML profile has no facility to specify routing rules at design time. Supporting tools transform a `MessageSplitter` to a skeleton source code (e.g. in Java) or rule description (e.g. in XPath) that performs message routings. Application developers complete the skeleton code/description. In an example model shown in Figure 1, a `MessageSplitter` divides a request message (`SpaceImage`) into two fragments, and sends them to different destinations (`AstronomyArchive` and `IndexingService`).

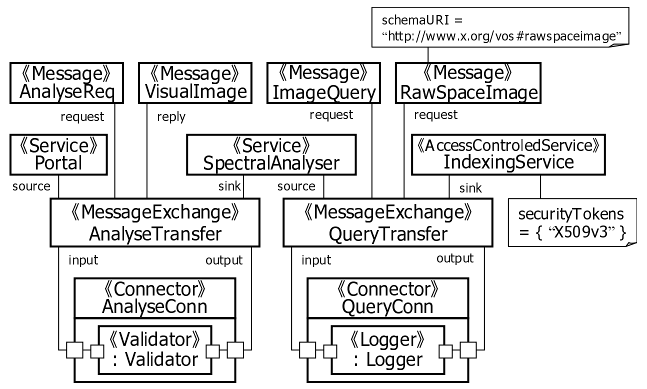


Figure 5. Astrophysical Virtual Observatory System

3.3. Service

`Service` has four optional tagged-values (Figure 6). `priority` is the priority of each message that a service issues. The range of `priority` is 0 to 255. (0 is the lowest and 255 is the highest.) Message queues (Section 3.1) and manycast filters (Section 3.2) use this value.

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

`redundancy` specifies the number of runtime instances of a service. Redundant services are used to implement fault tolerance and load balancing with manycasting and anycasting, respectively (Section 3.2).

`securityTokens` specifies security tokens (or certificates) required for a service to authenticate the source (service) of each incoming message. This tagged-value can contain multiple values in order of precedence. The values use the names defined in the WS-SecurityPolicy specification [14]. For example, in Figure 5, an `IndexingService` requires an X.509 certificate. (X.509 certificate is used if a message sender gives both security tokens.)

`AccessControlledService` is a stereotype extending the stereotype `Service`. It is a special type of service that enforces an access control policy. `securityTokens` must be specified in this service for the purpose of authentication. In Figure 5, an `IndexingService` controls accesses from a `SpectralAnalyser` using X.509 certificates. 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.

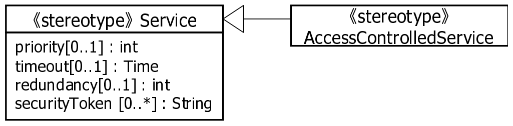


Figure 6. Tagged-Values of Service

3.4. Message

`Message` has a mandatory tagged-value, `schemaURI`, and three optional tagged-values: `priority`, `timeout` and `signatureMethod` (Figure 7). `schemaURI` identifies the schema of messages as a URI. Application designers can arbitrarily define it in their applications. For example, they may define http://www.x.org/vos#rawspaceimage as the URI to represent the schema of space image messages in Figure 5.

In addition to `Message` and `Connector`, `Service` also has tagged-values `priority` and `timeout` (Sections 3.1 and 3.3). They are used to specify the priority of a message and the timeout period of a message transmission between a source to destination, respectively. The precedence is that `Message`'s tagged-values override `Service`'s ones, which override `Connector`'s ones.

In order to ensure the integrity of a message, `signatureMethod` is used to specify an algorithm for generating the message's signature. The algorithm is represented with a URI defined in the XML Signature specification [25]. For example DSA (Digital Signature Algorithm) is represented with http://www.w3.org/2000/09/xmldsig#dsa-sha1. In Figure 1, each `SpaceImage` message is signed with DSA. When `signatureMethod` is specified, each message maintains its signature in `signature`.

The stereotype `EncryptedProperty` is used to specify encryption applied to a message. This stereotype is attached to data fields to be encrypted in a message class (see Figure 1). `EncryptedProperty` is defined as a stereotype extending `Property` in the UML metamodel. It has a tagged-value, `algorithm`, to specify an algorithm used to encrypt a message. The semantics of this tagged-value is same as that of `encryptionAlgorithm` in `Connector` (Section 3.1). An encryption algorithm is specified as a URI that the XML Encryption specification defines [24]. Different data fields in a message can be encrypted with different encryption algorithms. In Figure 1, `metaData` in `SpaceImage` is encrypted with Triple DES, which is represented with http://www.w3.org/2001/04/xmlenc#tripledes-cbc.

4. Model-Driven Development of Service Oriented Grid Applications with Ark

This section demonstrates a model-driven development (MDD) tool, called Ark, which accepts a UML model designed with the proposed UML profile and generates appli-

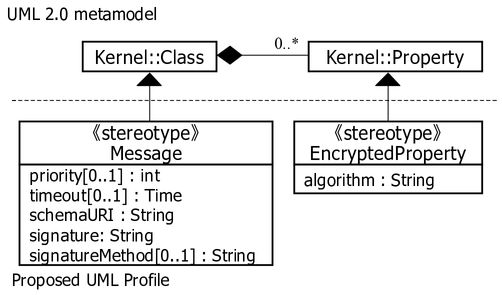


Figure 7. Tagged-Values of Message

cation code (source code and deployment descriptors). Currently, Ark implements two types of mappings from the proposed UML profile to MuleESB [2] and GridFTP [3]. Application designers give their UML models to the MDD tool, and instruct Ark which mapping to use for transforming their UML models to application code.

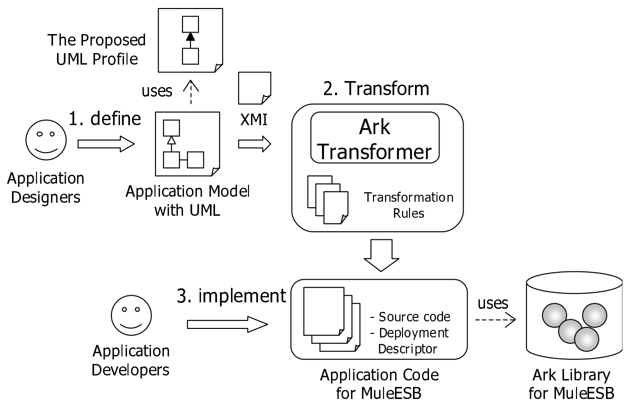


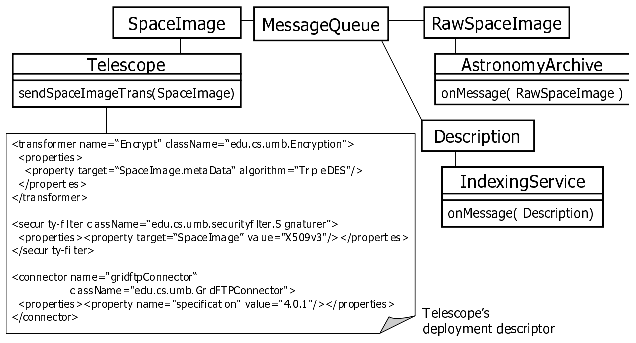
Figure 8. Application Development with Ark

Figure 8 shows the development process using Ark when MuleESB is used to as middleware to operate applications. Application designers define an application model with UML (e.g., example models in Figures 1 and 5) using the proposed UML profile. The Ark transformer, one of the components in Ark, takes the application model in the format of the XML Metadata Interchange (XMI) and transforms the input model into Java code compliant with MuleESB. Ark has been tested with MagicDraw, a visual UML modeling tool that can serialize UML models to XMI. Ark Transformer is implemented based on AndroMDA, a model transformation engine. Mapping rules in Ark is implemented as AndroMDA transformation templates, which defines how to transform UML model elements to application code elements.

In mapping rule from the proposed UML profile to MuleESB, a UML class stereotyped with ≪`Message`≫ or ≪`Message`≫ is mapped to a Java class with the same class name and the same data fields (Figure 9). Ark inserts several operations to the Java class corresponding to a service, depending on whether its association role is `source`/`sink` against a message exchange. The operations are used to send and receive messages: `sendX()` to send messages where `X` references the name of a message exchange, and `onMessage()` to receive messages. Fragment of the mapping rule is shown as follows. `$service` represents a class stereotyped with ≪`Service`≫. `$sourceMessageExchange`, `$requestMessage` and `$replyMessage` represent a `MessageExchange` identified with `source`, a request and a reply message respectively. Ark replaces each variable in the template with names (e.g., class and package names) specified in a UML model, and generate Java code. When a service has multiple requests or replies, a set of `sendX()` or `onMessage()` corresponding to them are generated.

```
package $service.packageName;
public class $service.name {
  public void send${sourceMessageExchange.name}(
    $requestMessage.fullyQualifiedName request){
    // initialize message transmission
    // send a message using MuleESB's API
  }
  public void onMessage(
    $replyMessage.fullyQualifiedName reply){
  }
}
```

The message transmission/processing semantics specified in a UML model is implemented in Java classes of message sender and destination. For example, in Figure 1, an `Telescope` sends a `SpaceImage` message to `AstronomyArchives`/`IndexingService` in an oneway manner. Therefore, Ark generates program code to send the message unidirectionally using MuleESB's API, and embeds the code in `sendSpaceImageTransfer()` of `Telescope`. In addition to Java classes, a deployment descriptor is generated to initialize an application (e.g., to deploy services and establish connections between them).



```
<transformer name="Encrypt" className="edu.cs.umb.Encryption">
  <properties>
    <property target="SpaceImage.metaData" algorithm="TripleDES"/>
  </properties>
</transformer>

<security-filter className="edu.cs.umb.securityfilter.Signaturer">
  <properties><property target="SpaceImage" value="X509v3"/></properties>
</security-filter>

<connector name="gridftpConnector"
           className="edu.cs.umb.GridFTPConnector">
  <properties><property name="specification" value="4.0.1"/></properties>
</connector>
```

Figure 9. Generated Code for MuleESB

Since MuleESB does not support several semantics

[2]http://mule.codehaus.org/. One of the major ESB implementations.
[3]An extension to FTP for transmitting large size of files [1].

that the proposed UML provides (e.g., queue and message signatures), Ark provides a set of components, called Ark library, which implements the missing semantics for MuleESB. Ark generates application code that uses components in Ark library (Table 1). After generation of application code, application developers complete their applications by, for example, writing method code.

In Figure 1, `BatchConn` is specified to work as a message queue and divide a `SpaceImage` message into two messages, `RawSpaceImage` and `Metadata`. Since MuleESB does not support a message queue, Ark implements a message queue using Java Message Service (JMS) and provides it as a service compliant with MuleESB. In Figure 9, the message queue (`MessageQueue`) is deployed between `Telescopes` and `AstronomyArchives`/`IndexingService`, and configured according to semantics specified in the model. The `MessageSplitter` is implemented as a class that implements the interface `org.mule.routing.outbound.AbstractMessageSplitter`. In MuleESB, the class can be attached to a service, and splits outgoing messages into fragments and routes them to different services.

As Figure 1 shows, the data field `metaData` in `SpaceImage` is encrypted. Since MuleESB does not support message-level encryption, Ark implements a pair of message transformers to encrypt and decrypt data fields in a message. In MuleESB, each service can have an arbitrary number of message transformers as the classes that implement the interface `org.mule.transformer.UMOTransformer`. Message transformers are invoked when a service receives a message or when it sends out a message. When a message is designed to use message-level encryption in an input UML model, a deployment descriptor is generated to configure the sender and receiver services of the message so that they use the message encryption/decryption transformers that the authors implemented. Figure 9 shows a fragment of generated deployment descriptor for `Telescope`. It instructs `Telescope` to use a message encryption transformer (`edu.cs.umb.Encryption`) to encrypt the data field `metaData` with the Triple DES when it sends out a `SpaceImage`.

As Figure 1 shows, each `SpaceImage` message is signed with DSA. Since MuleESB does not support DSA signatures, Ark provides a set of security filters to write/read the signatures and security tokens by implementing the interface `org.mule.umo.security.UMOEndpointSecurityFilter`. Similar to message transformers described above, security filters are invoked when a service receives a message or when it sends out a message. Ark generates deployment descriptor that configures services to use the security filters Ark provides. Figure 9 shows a fragment of generated deployment descriptor for `Telescope`. It configures `Telescope` to include a DSA signature in each `SpaceImage` message using a filter (`edu.cs.umb.securityfilter.Signature`).

In Figure 5, `IndexingService` performs authentication with X.509. As well as message signature, MuleESB does not support X.509 security tokens. Ark provides a security filter (`edu.cs.umb.securityfilter.SecurityToken`) to add a security token to messages, and it configures services (`SpectralAnalyse`) to use the filter.

Since MuleESB does not support the semantics of delivery assurance described in Section 3.1, Ark implements a pair of message interceptors to support the semantics. Each interceptor is implemented as a service in MuleESB. One of interceptors, called sender-side interceptor, intercepts messages right after a service send them, embeds an unique ID and a timestamp into each message, stores the messages on memory, and forwards them to their destinations. The other interceptor, called receiver-side interceptor, intercepts messages right before a service receive them, and sends back an Ack with IDs embedded in the messages to the sender-side interceptor. It allows the sender-side interceptor to know which messages are delivered and the sender-side interceptor can resend undelivered messages when `ExactlyOnce` is specified. Also, the receiver-side interceptor can detect if the same message is delivered more than once by checking its ID when `AtMostOnce` is specified. In addition to that, the receiver-side interceptor can sort messages by its departure time by using their timestamps when `InOrder` is specified.

In this astrophysical virtual observatory system, `Telescopes` employ GridFTP as a message transmission protocol to send a `SpaceImage` to a `BatchConn` (a message queue). Also, GridFTP is used to transmit a `RawSpaceImage` to a `AstronomyArchive` (Figure 1). Although MuleESB does not support GridFTP, it provides a plug-in mechanism to implement arbitrary message transmission protocols. Ark implements a plug-in for GridFTP (`edu.cs.umb.GridFTPConnector`) so that services can use it in MuleESB. As shown in Figure 9, Ark transformer generates a deployment descriptor configureing `Telescope` to use the GridFTP plug-in to transmit `SpaceImage` messages.

5. Related Work

There are several UML profiles proposed for SOA. [18] and [13] propose UML profiles to specify functional aspects in SOA. Both profiles are defined based on the XML schema of Web Service Description Language (WSDL). Each of the profiles provides a set of stereotypes and tagged-values that correspond to elements in WSDL, such as

Table 1. Components in Ark Library

Component	Description
Message Queue	Achieve connector's queue semantics. Implemented as a service using JMS.
Message Assurance	Achieve the semantics of delivery assurance. Implemented as a set of services.
Message Encryption Filter	Encrypt/Decrypt message's properties. Implemented as a set of security filters in MuleESB
Message Signature Filter	Add a signature to messages. Implemented as a security filter.
Security Token Filter	Add a security token to messages. Implemented as a security filter.
GridFTP Protocol Plug-in	Provide GridFTP as a message transmission protocol. Implemented as a plug-in for a protocol in MuleESB.

`Service`, `Port`, `Messages` and `Binding`[4]. Since WSDL is designed to define only functional aspects of web services, non-functional aspects are beyond of the scope of [18] and [13]. The proposed profile focuses on specifying non-functional aspects in SOA.

[2] proposes a UML profile to describe both functional and non-functional aspects in SOA. The stereotypes in this profile are generic enough to specify a wide range of applications. However, their semantics tend to be ambiguous. For example, the stereotypes for non-functional aspects include ≪`policy`≫, ≪`permission`≫ and ≪`obligation`≫, and ≪obligation≫ is intended to specify the responsibility of a service. [2] does not precisely define what developers have to (or can) specify with this stereotype and how to represent service responsibility (e.g. using natural languages or parameter values). In contrast, the proposed profile carefully defines its stereotypes and tagged-values in an unambiguous manner so that supporting tools can interpret and transform models to code.

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

[11] proposes a UML profile to describe dynamic service discovery in SOA. This profile provides a set of stereotypes such as ≪`uses`≫, ≪`requires`≫ and ≪`satisfies`≫ to specify relationships among service implementations, service interfaces and functional requirements. For examples, users can specify relationships in which a service *uses* other services, and a service *requires* other services that *satisfy* certain functional requirements. These relationship specifications are intended to effectively aid dynamic discovery of services. The proposed profile and [11] focus on different issues in SOA. Service discovery is beyond of the scope of the proposed profile, and [11] does not consider non-functional aspects in message transmission, message processing, security and service deployment.

[9], [12] and [20] define UML profiles to specify service orchestration in UML and map it to Business Process Execution Language (BPEL). 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.

[17] proposes a UML profile, called SecureUML, to define role-based access control for network applications. SecureUML provides notations to assign roles to classes (≪`security.role`≫), and notations to define access control permissions (≪`security.constraint`≫). SecureUML employs Object Constraint Language (OCL) to define access control. [15] proposes another UML profile, called UMLsec, to define data encryption (≪`data security`≫) and secure network links (≪`encrypted`≫). These UML profiles are parallel to the proposed profile in terms of the ability to describe security aspects in network applications. However, the proposed UML profile allows application developers to specify not only security aspects but also many other non-functional aspects (e.g., message transmission and message processing) required in SOA.

There are several research efforts to investigate implementation techniques for non-functional aspects in SOA [3, 19, 27, 23]. Each technique provides a means to implement non-functional requirements in, for example, performance, reliability and security and to enforce services to follow the requirements. Rather than providing specific implementations of non-functional aspects in SOA, the proposed UML profile is intended to provide a means for users to model and maintain non-functional aspects in vitual and implementation independent manners so that they can be mapped on different implementation technologies.

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

6. Conclusion

This paper proposes a UML profile to graphically specify and maintain non-functional aspects in SOA without depending on specific implementation technologies. This paper presents design details of the proposed profile, and describes how it is used in model-driven development of service-oriented grid applications.

7. Acknowledgement

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

References

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, C. D. M. Link, I. Raicu, and I. Foster. The Globus Striped GridFTP Framework and Server. *Proc. of Super Computing*, November 2005.

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

[3] F. Baligand and V. Monfort. A Concrete Solution for Web Services Adaptability Using Policies and Aspects. *Proc. of International Conference on Service Oriented Computing*, December 2004.

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

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

[6] T. Eri. *Service-Oriented Architecture: Concepts, Technology and Design*. Prentice Hall, 2005.

[7] I. Foster. Service-Oriented Science. *Science*, 308(5723), May 2005.

[8] L. Fuentes and A. Vallecillo. An Introduction to UML Profiles. *The European Journal for the Informatics Professional*, 2(5), April 2004.

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

[10] W. GuiLing, L. YuShun, Y. ShengWen, M. ChunYu, X. Jun, and S. MeiLin. Service-Oriented Grid Architecture and Middleware Technologies for Collaborative E-Learning. *Proc. of IEEE International Conference on Services Computing*, July 2005.

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

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

[13] IBM. *UML 2.0 Profile for Software Services*. developerWorks (online), April 2005.

[14] R. S. IBM, Microsoft and VeriSign. Web Services Security Policy Language, December 2002.

[15] J. Jїrjens. UMLsec: Extending UML for Secure Systems Development. *Proc. of ACM/IEEE International Conference on Unified Modeling Language*, October 2002.

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

[17] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. *Proc. of ACM/IEEE International Conference on Unified Modeling Language*, October 2002.

[18] E. Marcos, V. de Castro, and B. Vela. Representing Web services with UML: A Case Study. *Proc. of the International Conference on Service Oriented Computing*, December 2003.

[19] N. Mukhi, R. Konuru, and F. Curbera. Cooperative Middleware Specialization for Service Oriented Architectures. *Proc. of ACM International World Wide Web Conference*, 2004.

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

[21] Object Management Group. UML 2.0 Superstructure Specification, July 2005.

[22] S. Paunov, J. Hill, D. C. Schmidt, J. Slaby, and S. Baker. Domain-Specific Modeling Languages for Configuring and Evaluating Enterprise DRE System Quality of Service. *Proc. of IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, March 2006.

[23] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(4), February 2006.

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

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

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

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