Early Aspects for Non-functional Properties in Service Oriented Business Processes

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

In Service Oriented Architecture, each application is often designed with a set of reusable services and a business process. In order to retain the reusability of services, it is important to separate non-functional properties of applications (e.g., security and reliability) from their functional properties. Currently, non-functional properties are often defined on a per-service basis. In contrast, this paper investigates a new per-process strategy, and proposes an aspect oriented language to separate functional and non-functional properties in business processes. Each aspect formally specifies non-functional properties that crosscut among multiple services. The proposed language frees applications developers from manually specifying and validating non-functional properties for services one by one, thereby reducing the burdens/costs of application development and maintenance. This paper describes the design of the proposed language and demonstrates how each aspect (i.e., a set of non-functional properties) is woven to a business process and transformed to application code.

1. Introduction

Service Oriented Architecture (SOA) is an emerging style of software architectures to build, integrate and maintain applications in a cost effective manner by improving their reusability [1,2]. In SOA, each application is often designed in an implementation independent manner with a set of reusable *services* and a *business process*. Each service encapsulates the function of an application component, and each business process defines how services interact to accomplish a certain business goal. Services are intended to be reusable (or sharable) for different applications to implement different business processes.

For retaining the reusability of services, it is important to separate non-functional properties of applications (e.g., security and reliability) from their functional properties because different applications use each service in different non-functional requirements (e.g., different security policies) [3–5]. In most of the current practices of separating functional and non-functional properties in SOA, non-functional properties are specified on a per-service basis [4–9]. However, with this *per-service* strategy, application developers need to manually ensure that each non-functional property is properly configured in a series of services in an ad-hoc manner because each non-functional property tends to cover multiple services simultaneously. For example, a certain security property may be applied to all services that participate in a business process. It is tedious, expensive and error-prone to consistently specify and validate non-functional properties throughout services in a large-scale business process.

In order to address this issue, it is necessary to specify non-functional properties for business processes rather than services. A *per-process* strategy can free application developers from manually specifying and validating non-functional properties for services one by one, thereby reducing the burdens/costs of application development and maintenance. However, this strategy has not been supported yet in the current visual/textual definition languages for business processes, such as Unified Modeling Language (UML), Business Process Modeling Notation (BPMN) [10] and Business Process Execution Language (BPEL) [11].

This paper proposes a language for a new per-process strategy to separate functional and non-functional properties in SOA. The proposed language leverages the notion of *aspects* (or *early aspects*[1]) in aspect oriented programming/modeling [13, 14]. Each aspect is used to specify non-functional properties that crosscut (or scatter) over multiple services in a business process.

Figure 1 shows a model-driven development (MDD) framework that supports the proposed aspect oriented language. The framework consists of (1) the proposed language, (2) a UML profile to specify non-functional proper-

[1]Early aspects are crosscutting concerns that exist in early phases in application development process, such as requirement analysis phase [12].

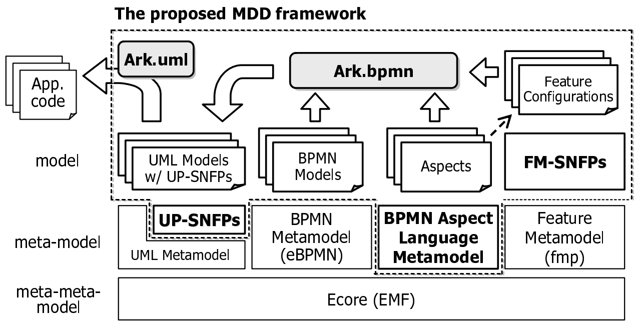


Figure 1: An MDD Framework supporting the Proposed Aspect Oriented Language

ties in SOA, called, UP-SNFPs [4], (3) a feature model that defines a set of constraints among non-functional properties (e.g. dependency and mutual exclusion constraints), called FM-SNFPs [5], (4) a model transformation tool, called called Ark. All artifacts in this framework are maintained with the metameta model (Ecore) of the Eclipse Modeling Framework (EMF[2]). The proposed language's syntax is defined as a meta model on Ecore. UP-SNFPs is defined as an extension to the UML metamodel. FM-SNFPs is defined on the feature metamodel in fmp[3]. Currently, BPMN is used as a language to define business processes. BPMN models are defined on eBPMN[4]. Ark consists of two components: Ark.bpmn and Ark.uml. Ark.bpmn interprets a given aspect (i.e., a set of non-functional properties), waive it to a BPMN model, and transform the BPMN model to a UML model decorated with UP-SNFPs. Ark.uml transforms the generated UML model into application code (program code and deployment descriptors).

Using the proposed aspect oriented language, non-functional properties can be specified for business processes in an implementation independent manner. They can be portable and reusable across different implementation technologies. Through a chain of model transformations, Ark generates application code specific to certain implementation technologies such as Enterprise Service Buses (ESBs).

2. Preliminaries

This section overviews BPMN (Section 2.1) and describes UP-SNFPs and FM-SNFPs, which are used in application development with (Sections 2.2 and 2.3).

2.1. BPMN

BPMN is a visual language to define business processes. Figure 2 shows a purchasing process that involves three services: `Customer`, `Retailer` and `Supplier`. When a retailer receives an order, it examines whether it has ordered items in stock. If not, it places an order to a supplier. The supplier ships ordered items or notifies a shipping delay. A retailer asks a customer to give feedback one week after a payment.

[2]http://www.eclipse.org/emf/
[3]http://gp.uwaterloo.ca/fmp/
[4]http://www.soyatec.com/ebpmn/

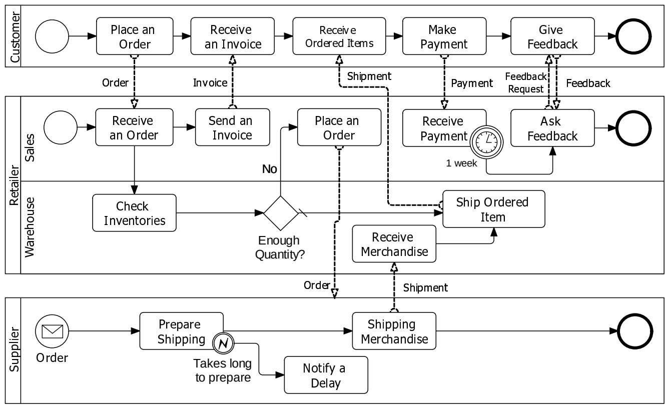


Figure 2: A Purchasing Process

A BPMN model consists of *Pools*, *Tasks* and *Sequence/Message Flows*. A Pool, represented as a rectangle, denotes a participant in a business process; for example, `Retailer` in Figure 2. A Pool can have *Lanes* to classify internal activities; for example, `Sales` and `Warehouse` in `Retailer`. A Task, represented as a rounded-corner rectangle, denotes a task performed by a participant; for example, `Receive an Order` in `Retailer`. A *Sequence Flow*, represented as a solid line, denotes the order of Tasks performed in a Pool. A *Message Flow*, represented as a dashed line, denotes a flow of messages between two participants.

A Pool can contain *Gateways* and *Events*. A Gateway, represented as a diamond, controls the divergence (forking) and convergence (merging) of Sequence Flows. A Gateway can have the default Sequence Flow, represented as a Sequence Flow with a slash mark, which denotes a Sequence Flow that is chosen if others are not selected. In Figure 2, a `Retailer` branches a flow depending on the number of inventories. If it does not have enough inventories, it performs `Place an Order`. If it does, it selects the default Sequence Flow and performs `Ship Ordered Item`.

An Event, represented as a circle, triggers a subsequent Sequence Flow. BPMN supports several types of events: *Message*, *Timer*, *Rule*, *Error*, *Cancel* and *Compensation*. A Message, represented as a circle with an envelope, denotes a reception of a message from another participant. In Figure 2, a `Supplier` triggers its process when it receives an `Order` message from `Retailer`. A Timer, represented as a circle with a clock, denotes a specific time or interval. In Figure 2, a `Retailer` performs `Ask Feedback` one week after it performs `Receive Payment`. An Error, represented as a circle with a lightning, denotes a specific error condition. In Figure 2, a `Supplier` performs `Notify a Delay` when `Prepare Shipping` takes a long time.

2.2. UP-SNFPs

UP-SNFPs is a UML profile to visually specify non-functional properties in UML's class and composite structure diagrams [4]. It is designed around two major con-

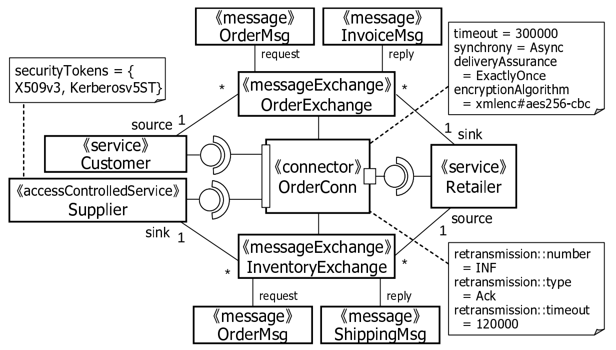

《message》 OrderMsg
《message》 InvoiceMsg
request
reply
timeout = 300000
synchrony = Async
deliveryAssurance = ExactlyOnce
encryptionAlgorithm = xmlenc#aes256-cbc
securityTokens = { X509v3, KerberosV5ST}
《messageExchange》 OrderExchange
source
sink
《service》 Customer
《connector》 OrderConn
《service》 Retailer
《accessControlledService》 Supplier
sink
source
《messageExchange》 InventoryExchange
retransmission::number = INF
retransmission::type = Ack
retransmission::timeout = 120000
request
reply
《message》 OrderMsg
《message》 ShippingMsg

Figure 3: An Example Model with UP-SNFPs

cepts in SOA: *services* and *connections* between services. Each connection defines how services are connected with each other and how messages are exchanged through the connection. UP-SNFPs covers the following four areas of non-functional properties.

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

Figure 3 shows an example UML model defined with UP-SNFPs. It illustrates a purchasing application, which corresponds to the BPMN model in Figure 2. In this example, three services (Customer, Retailer and Supplier) exchange messages. Each service is represented by a class stereotyped with ≪service≫ or ≪accessControlledService≫. ≪accessControlledService≫ indicates a special type of services that enforce an access control policy. The tagged-value securityTokens is used to specify security tokens (or certificates) for access authentication.

In Figure 3, services exchange three types of messages, each of which is stereotyped with ≪message≫. Each pair of a request and reply messages is represented by a class stereotyped with ≪messageExchange≫. For example, a pair of OrderMsg (request) and InvoiceMsg (reply) is represented by OrderExchange in Figure 3.

≪connector≫ represents a connection that transmits messages between services. In Figure 3, messages are delivered through the connector 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 (a ball icon) and a required interface (a socket icon). Services use the provided and required interfaces to send and receive messages, respectively. In Figure 3, a Customer sends an OrderMsg to a Retailer.

Each connector can have multiple tagged-values to specify a set of message transmission and processing semantics. In Figure 3, the connector OrderConn specifies the timeout of message transmissions (300,000 milliseconds), synchrony of message transmissions (asynchronous), assurance level of message delivery (exactly once), and encryption algorithm for messages (Advanced Encryption Standard).

2.3. FM-SNFPs

There often exist a number of constraints (e.g., dependency and mutual exclusion) among non-functional properties. In UP-SNFPs, a timeout period must be specified with the timeout tagged-value when the synchrony of message transmissions is configured as asynchronous (synchrony=Async; see Figure 3). A message retransmission policy requires to specify the maximum number of retransmissions and its type: ack-based or nack-based. If it is configured as ack-based, a timeout period must be specified.

Following the notion of feature modeling [15], FM-SNFPs provides a feature model that explicitly defines non-functional constraints in SOA [5]. Feature modeling is a simple yet powerful method to specify a set of constraints among an application's *features* (e.g., configuration policies). By modeling a non-functional property as a feature, FM-SNFPs allows developers to consistently validate and enforce non-functional constraints in their applications.

Figure 4 shows a subset of the feature model in FM-SNFPs. FM-SNFPs has a hierarchy of non-functional properties. White and black circle icons indicate optional and mandatory non-functional properties, respectively. In Figure 4, Timeout is optional. A fork icons with white and black sectors denote *exclusive-OR* and *OR* relationships among non-functional properties, respectively. In Figure 4, only one of Sync, Async or Oneway must be selected for Synchrony. A requires relationship indicates a dependency among non-functional properties. For example, when Async and Retransmission are selected, Type and Timeout must be selected too. An encourages relationship has a similar but weaker semantics than a requires relationship. For example, when Access Control is selected, it is encouraged (but not mandatory) to select Message Encryption too. Figure 5 shows an example feature configuration (an instance of the FM-SNFPs feature model). It shows a set of non-functional properties selected for a certain application.

3. The Proposed Aspect Oriented Language

In general, aspect oriented languages are designed to separate crosscutting concerns from other concerns and modularize crosscutting ones as an aspect [13, 14]. Then,

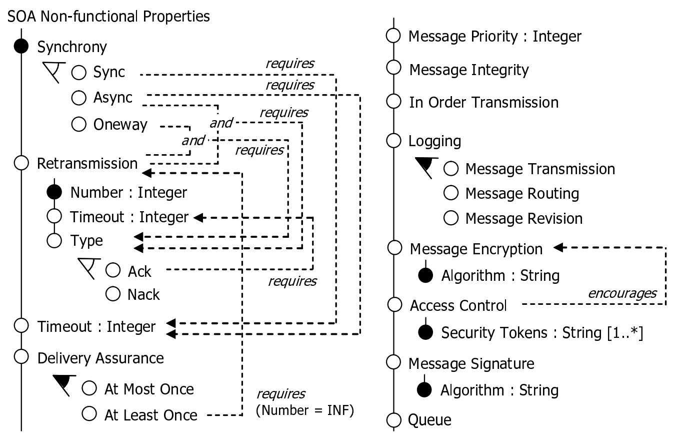


Figure 4: The Definition of FM-SNFPs

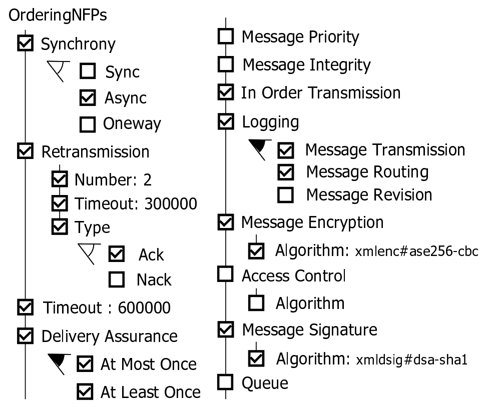


Figure 5: An Example Feature Configuration

supporting tools (often called *aspect weavers*) weave aspects into the other parts of an application to complete it.

As described in Section 1, non-functional properties are crosscutting concerns. Thus, the proposed aspect oriented language is designed to identify a set of non-functional properties used in a business process, modularize them as an aspect and instruct how they are woven to the business process. Ark.bpmn serves as an aspect weaver for the proposed language (see also Figure 1). This clear separation between functional and non-functional properties allows the two types of properties to evolve in parallel, thereby improving the maintainability of applications.

An aspect consists of *advice*s and *pointcut*s. An advice is a definition (or implementation) of a concern that appears many places (e.g., logging code) and a pointcut specifies places where advices appear (e.g., at the beginning of methods). Supporting tools insert (or weave) advices into main application code according to pointcuts. The proposed aspect oriented language treats a feature configuration that specifies a set of non-functional properties as an advice. Also, the proposed aspect oriented language provides crosscutting expressions that allow for developers to define pointcuts against BPMN models. Ark weaves non-functional properties into BPMN models according to pointcuts.

Listing 1 is a definition of an aspect, which begins with the keyword `aspect` followed by the name of an aspect, in the proposed aspect oriented language. The aspect `OrderNFPAspect` defines arbitrary number of pointcuts with the keyword `pointcut` followed by the name of a pointcut. The pointcut `order` specifies paths between two model elements (Task or Message Flow) in a BPMN model by using the `within` join point[5] (Line 3 and 4). The `within` join point takes two names of model elements in regular expression as parameters. A set of model elements matched to the first parameter and the second parameter are considered as starting points and ending points of paths respectively. Ark finds paths between every combinations of starting and ending points by following Sequence or Message Flows in a given BPMN model. Since a pair of two points may be connected with multiple paths, a supporting tool follows all Sequence and Message Flows from a starting point to find ending points as Listing 2 shows.

Listing 1: An Example Aspect

```
1  aspect OrderNFPAspect{
2   // pointcut
3   pointcut order:
4    within("Customer::Place .*", "Customer::.*Items");
5
6   // advice
7   order: OrderingNFPs; }
```

Listing 2: Path Search Algorithm

```
// corresponds to a join point "within(startName, endName)"
findPaths(startName, endName){
 // finds Tasks or Message Flows by names
 startPoints = findElementsByName(startName)
 endPoints   = findElementsByName(endName)

 foreach(s in startPoints){
  // find paths from 's' to 'endPoints'
  paths += _findPaths(s, endPoints, [])
 }
 return paths;
}

_paths = []
_findPaths(startPoint, endPoints, pathSoFar){
 pathSoFar.add(startPoint)
 if startPoint is included in endPoints
  _paths.add(pathSoFar)
 else
  // check all points next to startPoint
  // by following all outgoing Sequence/Message Flows
  foreach(nextPoint in startPoint.outgoings){
   _paths.add(
    _findPaths(nextPoint, endPoints, pathSoFar))
  }
  return _paths
}
```

The pointcut `order` (Line 3 in Listing 1) specifies paths, consisting of Tasks and Message Flows, between two model elements: `Customer::Place .*` and `Customer::.*Items` (Line 4). As Figure 2 shows, these two elements (parameters) match the tasks `Place`

[5]A join point is a place where advices can be woven. A pointcut is a set of join points.

an `Order` and `Receive Ordered Items` in `Customer` respectively, and there are four paths between them. The first path starts with `Customer::Receive Ordered Items`, passes through `Customer::Receive an Invoice` and reaches `Customer::Receive Ordered Items`. The second path starts with `Customer::Receive Ordered Items`, passes through `Order`, `Retailer::Receive an Order`, `Retailer::Send an Invoice`, `Invoice` and `Customer::Receive an Invoice`, and reaches `Customer::Receive Ordered Items`. The third path starts with `Customer::Receive Ordered Items`, passes `Retailer::Receive an Order`, `Retailer::Check Inventories` and `Customer::Ship Ordered Item`, and reaches `Customer::Receive Ordered Items`. (Message Flows between Tasks are also included.) The last path passes through `Retailer::Place an Order` and Tasks in `Supplier` after `Retailer::Check Inventories`, and reaches `Customer::Receive Ordered Items`. As illustrated in this example, Ark automatically finds a set of model elements that involved in a (sub) business process according to pointcuts.

Then, Ark weaves a set of non-functional properties into model elements according to an advice (Line 7). The name of a pointcut (`order` in Line 3 and 7) refers to a set of model elements included in paths, and the name of an feature configuration (`OrderingNFPs` in Line 7) refers to a set of non-functional properties defined in a feature configuration (Figure 5). According to the advice, Ark weaves a set of non-functional properties (`OrderingNFPs`) into a set of model elements (`order`) in a consistent manner.

In addition to `within`, the proposed aspect oriented language supports several join points (Table 1): `target`, `source`, `flow`, `trigger`, `depth` and `default`.

Table 1: Join Points

Join Point	Description
within	Returns all paths between two model elements (Tasks or Message Flows).
target	Returns all paths arrive at certain services.
source	Returns all paths depart from certain services.
flow	Specifies certain Message Flows.
trigger	Returns all paths start from a certain type of event
depth	Limits the number of services in paths.
default	Follows only default Sequence Flows at Gateways

Listing 3: An Example Aspect

```
1  aspect NFPAspect{
2   // pointcuts
3   pointcut wholeProcess:
4    within("Customer::Place .*", ".*");
5
6   pointcut toRetailer: target("Retailer");
7
8   pointcut payment: flow("Payment");
9
10  pointcut error: trigger(ERROR);
11
12  pointcut orderWithRetailer:
13   within("Customer::Place.*", "Customer::.*Items") &&
14   depth(1);
15
16  pointcut retailerProcess:
17   within("Retailer::Receive an Order", ".*") &&
18   default();
19
20  pointcut feedback:
21   trigger(TIMER) &&
22   within(".*", "Customer::Give Feedback");
23
24  // advices
25  wholeProcess: DefaultSecurity, NoDeliveryAssurance;
26  orderWithRetailer: OrderingNFPs;
27  toRetailer: MessgeEncryption;
28  payment: HighlevelSecurity;
29  error: DeliveryAssurance; }
```

`target` and `source` are another representations of `within( ".*", "ServiceName::.*" )` and `within( "ServiceName::.*", ".*" )`. (`ServiceName` is a parameter of `target` and `source`.) They return all paths that arrive at or depart from certain services respectively. Listing 3 defines the pointcut `toRetailer` (Line 6) that selects paths arrive at `Retailer`. `target` and `source` allow for defining aspects that certain services requires. For example, when the `Retailer` requires all incoming/outgoing Message Flows to be singed and encrypted, developers can enforce the rule by using `target` and `source`.

`flow` is another representation of `within( "FlowName", "FlowName" )`. (`FlowName` is a parameter of `flow`.) It directly specifies certain Message Flows, and returns paths that contain only Message Flows. Listing 3 defines the pointcut `payment` (Line 8) that selects the Message Flow `Payment` in Figure 2. `flow` allow for defining aspects that certain Message Flow requires. For example, when the `Payment` Message Flow requires a high security level and an access control but others do not, developers use `flow` to specify non-functional properties of `Payment` directly.

`trigger` returns all paths that start from a certain type of event, i.e., Message, Timer, Rule, Error, Cancel or Compensation. Listing 3 defines the pointcut `error` (Line 10) that selects paths start from Error events. It allows for specifying non-functional properties for error handling processes (e.g., ensuring delivery assurance). `trigger` is transformed into `within` as well as `target`, `source` and `flow`. First, it finds all events of a certain type and uses them as the first parameter of `within` as `within( events, ".*" )`.

`depth`, which is used with other join points, specifies the number of services to be included in paths. For example, paths can contain three services when a pointcut has `depth(2)`. Listing 3 defines the pointcut `orderWithRetailer` using `depth` (Line 12 to 14). Although the `within` part is the same as that of `order` in Listing 1, the pointcut `orderWithRetailer` returns paths that involve only `Customer` and `Retailer` because of `depth`. `depth` can limit the range of interactions among services. For example, `orderWithRetailer` finds interactions that occur only among `Customer` and other services.

`default`, which is used with other join points as well as `depth`, selects paths containing default Sequence Flows at Gateways. Listing 3 defines the pointcut `retailerProcess` (Line 16 to 18) that selects a default Sequence Flow at a Gateway.

These join points can be used together. For example, Listing 3 defines the pointcut `feedback` (Line 20 to 22) that selects paths starting from Timer events and ending with the `Customer::Give Feedback` Task. The pointcut returns a path contains `Retailer::Ask Feedback`, `Feedback Request` and `Customer::Give Feedback`, but it does not contain the `Feedback` Message Flow because of `within( ".*", "Customer::Give Feedback")`. When a pointcut uses multiple join points, Ark returns an intersection of paths found by each join point. For example, the pointcut `feedback` returns a set of paths contained in both `trigger(TIMER)` and `within( ".*", "Customer::Give Feedback")`.

According to advices, non-functional properties defined in feature configurations are woven into a BPMN model. For example, Listing 3 weaves non-functional properties in `DefaultSecurity` and `NoDeliveryAssurance` into the pointcut `wholeProcess` (Line 24). Since it is the first advice appears in an aspect, Ark weaves the two sets of non-functional properties into a BPMN model first. Then, non-functional properties in `MessageEncryption` is woven into `toRetailer`. When non-functional properties in `DefaultSecurity` and `MessageEncryption` are contradict with each other (e.g., they specify different security level), `MessageEncryption` overwrites `DefaultSecurity` since `MessageEncryption` appears after `DefaultSecurity`.

This way, the proposed aspect oriented language separates BPMN models and its non-functional properties well and improves the reusability of feature configurations. For example, a feature configuration can be applied to all model elements in a certain business process as a default setting, or can be applied to only specific elements (e.g., elements in paths between certain services) by only changing pointcuts. It makes easy to configure applications in typical situations (e.g., services hosted in-house, or accessed via the Internet) by reusing existing feature configurations.

4. Application Development with Ark

Figure 6 shows the application development process with the proposed aspect oriented language. Ark.bpmn takes a BPMN model and an aspect(s), and transforms the BPMN model to a UML model defined with UP-SNFPs. (See also Figure 1.) Ark.uml transforms the generated UML model into a skeleton of application code.

Ark.bpmn performs a model transformation in two steps:

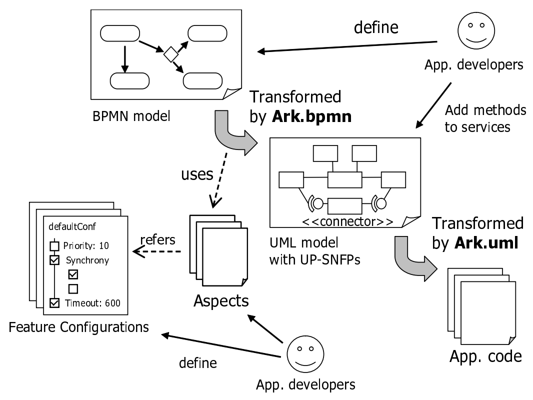


Figure 6: Development Process with Ark

(1) transforming a BPMN model into a plain UML model that defines no non-functional properties, and (2) configuring non-functional properties on the generated UML model based on the definition of an aspect(s).

The first step simply transforms a BPMN model into a UML model. The generated UML model does not have any non-functional properties, but it has several stereotypes defined in UP-SNFPs (e.g., ≪`service`≫ and ≪`connector`≫).

Figure 7 is a fragment of a UML model transformed from the BPMN model in Figure 2. (The UML model contains model elements corresponding to the `Customer` and `Retailer` Pools, and the `Order` and `Invoice` Message Flows in Figure 2.) Ark.bpmn transforms a Pool in a BPMN model into a class stereotyped with ≪`service`≫. (e.g., the `Customer` Pool is mapped into the Customer class.) Each Task in a Pool is transformed into a method in a class. (e.g., the `Place an Order` Task is mapped into the `placeAnOrder` method.) Also, a Message Flow among Pools is mapped into three classes: classes with ≪`connector`≫, ≪`messageExchange`≫ and ≪`message`≫. They represent a connector between services, a pair of a request and reply messages, and a request message[6] respectively. This transformation is implemented by leveraging UP-SNFPs' and eBPMN's meta-models.

The next step is to configure non-functional properties in a generated UML model according to aspects.

For defining aspects, Ark.bpmn provides an editor running on Eclipse. As Figure 8 shows, the editor shows built-in keywords in boldface, automatically performs a syntax check, and reports syntax errors while developers define aspects. The editor is implemented by leveraging oAW. oAW allows developers to define the syntax of user-defined languages in EMF (BPMN Aspect Language Metamodel in Figure 1), and it generates editors for the languages (Figure 8).

Ark.bpmn parses definitions of aspects and finds which

[6]Since a Message Flow in BPMN represents an oneway message, only a request message is generated in a UML model.

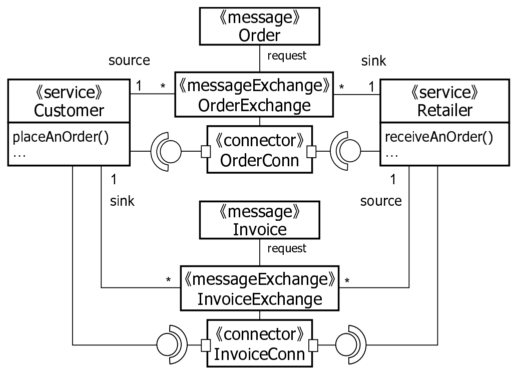
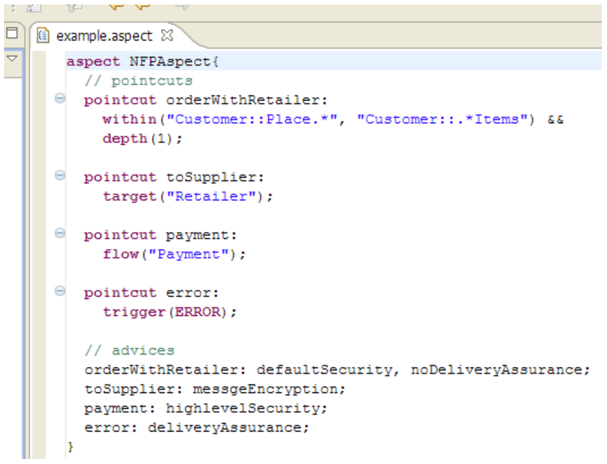


Figure 7: A Fragment of a generated UML model



```
aspect NFPAspect{
  // pointcuts
  pointcut orderWithRetailer:
    within("Customer::Place.*", "Customer::.*Items") &&
    depth(1);

  pointcut toSupplier:
    target("Retailer");

  pointcut payment:
    flow("Payment");

  pointcut error:
    trigger(ERROR);

  // advices
  orderWithRetailer: defaultSecurity, noDeliveryAssurance;
  toSupplier: messgeEncryption;
  payment: highlevelSecurity;
  error: deliveryAssurance;
}
```

Figure 8: An Editor in Ark.bpmn

feature configurations are applied to which model elements in a given BPMN model as described in Section 3. If a feature configuration is applied to certain Tasks and/or Message Flows in a BPMN model, the feature configuration is applied to services (classes with ≪service≫) that have methods corresponding to the Tasks and/or connectors (classes with ≪connector≫) corresponding to the Message Flows. For example, the feature configuration `OrderingNFPs` is applied to the pointcut `orderWithRetailer` in Listing 3 (Line 25). Since the pointcut `orderWithRetailer` returns paths that contain several Tasks in `Customer` and `Retailer` in a BPMN model (Figure 2), the feature configuration `OrderingNFPs` is applied to `Customer` and `Retailer` services in a generated UML model (Figure 7). Also, since the paths contains several Message Flows between `Customer` and `Retailer`, the feature configuration `OrderingNFPs` is applied to corresponding connectors in a generated UML model as well. Then, Ark.bpmn configures tagged-values defined in UP-SNFPs according to feature configurations.

Figure 9 is a UML model that Ark.bpmn generates by weaving the feature configuration in Figure 5 into the UML model in Figure 7.

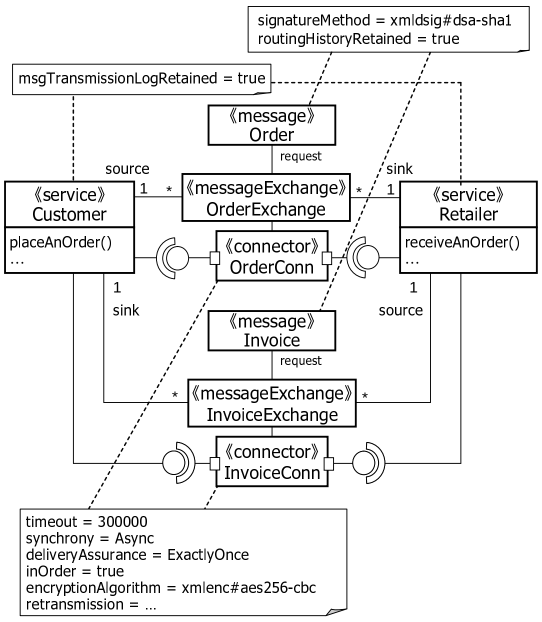


Figure 9: A UML model with Non-Functional Properties

Once Ark.bpmn completes its model transformation, Ark.uml validates an input UML model defined with UP-SNFPs against the UML metamodel and UP-SNFPs, and transforms it to a skeleton of application code (program code and deployment descriptors). (See Figures 1 and 6). Currently, Ark.uml implements a transformation mapping for two major ESBs, Mule ESB[7] and ServiceMix ESB[8], and GridFTP[9]. See [4] for full discussion on Ark.uml.

5. Related Work

This work is an extension to the authors' previous work [4, 5]. This work considers non-functional properties in business process models and proposes an aspect oriented language, while previous work considered them in UML models on a per-service basis.

AspectViewpoint is an aspect oriented language to define aspects for BPMN models [17]. It uses aspects to define business processes, and extends an existing business process by weaving the new ones to it. For example, a new process (e.g., cancellation process) may be defined as an aspect and woven into a purchasing process so that the purchasing process can consider the new one. [18] proposes an aspect oriented language to define aspects for BPEL. It uses aspects to define BPEL primitives (e.g., a branch of flows) and customize an existing business process by weaving the primitives to it. Although the above both languages consider aspects for business processes, they focus on functional properties of business processes (i.e., interactions among services) rather than non-functional properties.

[7]http://mule.codehaus.org/

[8]http://servicemix.apache.org/

[9]An extension to FTP for transmitting files of large size [16].

Unlike these languages, the proposed language focuses on non-functional properties in business processes.

AO4BPEL [19] is an aspect oriented language to extend BPEL business processes as [18] does. Unlike [18], it supports several non-functional properties such as reliable messaging, message encryption and transactions. Aspects can specify non-functional properties that are woven to services and their activities/tasks; however, the variety of pointcuts is limited in AO4BPEL. In contrast, the proposed aspect oriented language considers the pointcuts in control/message flows as well. This significantly increases its expressiveness. Also, it supports much more non-functional properties than AO4BPEL does. (See Section 2.)

[20] proposes a method to model and analyze non-functional requirements in business processes (e.g., desirable response time and throughput). It examines whether each service has conflicting non-functional requirements by inspecting which services involve in which business processes. However, [20] does not provide a formal language to weave non-functional requirements to business processes. The synthesis of functional and non-functional properties is manually performed. Code generation is not supported either. In contrast, the proposed aspect oriented language can formally define how to combine non-functional properties into business processes. Ark implements code generation for the proposed aspect oriented language.

[21] proposes a method to define a set of non-functional requirements as an aspect and weave it to a UML class model. However, it does not provide specific non-functional properties and does not perform code generation. In contrast, the proposed aspect oriented language defines aspects on a per-process basis, not on a per-service basis.

6. Conclusion

This paper proposes an aspect-oriented language for a new per-process strategy to separate functional and non-functional properties in SOA. Each aspect specifies non-functional properties that crosscut among multiple services in a business process. The proposed language frees applications developers from manually specifying and validating non-functional properties for services one by one, thereby reducing the burdens/costs of application development and maintenance. A supporting MDD tool interprets a given aspect (i.e., a set of non-functional properties), waive it to a BPMN model, transform the BPMN model to a UML model, and generate corresponding application code (program code and deployment descriptors).

References

[1] M. Bichler and K. Lin. Service-Oriented Computing. *IEEE Computer*, 39(6), June 2006.

[2] M. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *IEEE Int'l Conf. on Web Information Systems Engineering*, December 2003.

[3] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, and R. Shah. *Service-Oriented Architecture (SOA) Compass : Business Value, Planning, and Enterprise Roadmap*. IBM Press, October 2005.

[4] H. Wada, J. Suzuki, and K. Oba. A Model-Driven Development Framework for Non-Functional Aspects in Service Oriented Architecture. *Journal of Web Services Research*, September 2008. to appear.

[5] H. Wada, J. Suzuki, and K. Oba. A Feature Modeling Support for Non-Functional Constraints in Service Oriented Architecture. In *IEEE Int'l Conf. on Services Computing*, July 2007.

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

[7] G. Ortiz and J. Hernändez. Toward UML Profiles for Web Services and their Extra-Functional Properties. In *IEEE Int'l Conf. on Web Services*, September 2006.

[8] OASIS. *Web Services Security*, November 2006.

[9] OASIS. *Web Service Reliable Messaging, 1.1*, September 2004.

[10] Business Process Modeling Initiative. *Business Process Modeling Notation (BPMN) 1.0*, May 2004.

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

[12] AOSD-Europ Network of Excellence. *Extensive Survey of Aspect-Oriented Requirements Engineering, Architecture and Design Approaches*, May 2005.

[13] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *European Conf. on Object-Oriented Programming*, June 1997.

[14] T. Elrad, O. Aldawud, and A. Bader. Aspect-Oriented Modeling - Bridging the Gap Between Design and Implementation. In *ACM Int'l Conf. on Generative Programming and Component Engineering*, October 2002.

[15] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools and Applications*. Addison-Wesley, 2000.

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

[17] D. Correal and R. Casallas. Using Domain Specific Languages for Software Process Modeling. In *ACM OOPSLA Workshop on Domain-Specific Modeling*, October 2007.

[18] C. Courbis and A. Finkelstein. Weaving Aspects into Web Service Orchestrations. In *IEEE Int'l Conf. on Web Services*, October 2005.

[19] A. Charfi, B. Schmeling, A. Heizenreder, and M. Mezini. Reliable, Secure, and Transacted Web Service Compositions with AO4BPEL. In *IEEE European Conference on Web Services*, December 2006.

[20] F. Aburub, M. Odeh, and I. Beeson. Modelling non-functional requirements of business processes. *Information and Software Technology*, 49(11):1162–1171, November 2007.

[21] L. Xu, H. Ziv, D. Richardson, and Z. Liu. Towards Modeling Non-Functional Requirements in Software Architecture. In *ACM Int'l Conf. on Aspect-Oriented Software Development Early Aspects Workshop*, March 2005.