Leveraging Early Aspects in End-to-End Model Driven Development for Non-Functional Properties 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

In Service Oriented Architecture (SOA), 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. This paper investigates an end-to-end model-driven development framework that separates non-functional properties from functional properties and consistently manages them from high-level business processes to low-level implementation configurations. This framework proposes two key components: (1) a programming language, called BALLAD, for a new *per-process* strategy to specify non-functional properties for business processes and (2) a graphical modeling method, called FM-SNFPs, to define a series of constraints among non-functional properties. BALLAD leverages the notion of aspects in aspect oriented programming/modeling. Each aspect is used to specify a set of non-functional properties that crosscut multiple services in a business process. FM-SNFPs leverages the notion of feature modeling in order to explicitly define constraints among non-functional properties such as dependency and mutual exclusion constraints. BALLAD and FM-SNFPs can free application developers from manually specifying, maintaining and validating non-functional properties and constraints for services one by one, thereby reducing the burdens/costs in development and maintenance of service-oriented applications. This paper describes the design details of BALLAD and FM-SNFPs, and demonstrates how they are used in developing service-oriented applications. Empirical evaluation results show that BALLAD significantly reduces the costs to implement and maintain non-functional properties in service-oriented applications. BALLAD and FM-SNFPs are designed and implemented efficient and scalable.
Keywords: Non-Functional Properties, Model-Driven Development, Service Oriented Architecture, Aspect Oriented Software Development, Business Process Modeling, Feature Modeling

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 (Bichler & Lin, 2006; Papazoglou & Heuvel, 2007; Erickson & Siau, 2008). 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.

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 because different applications use each service in different non-functional contexts (e.g., different security policies) (?, ?; Bieberstein, Bose, Fiammante, Jones, & Shah, 2005). For example, an application may transmit 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 messages. Another application may transmit plain messages to the service when it is deployed in-house. Separation of functional and non-functional properties improves the reusability of services in different non-functional contexts.

This paper investigates end-to-end model-driven development (MDD) that manages non-functional properties from high-level business process modeling to low-level configurations of implementation technologies such as transport protocols and remoting middleware. To this end, there exist two major research issues: (1) a lack of adequate strategies to specify non-functional properties in business processes and (2) a lack of adequate methods to manage constraints among non-functional properties.

The first issue is regarding the strategies to specify non-functional properties in business processes. In most of the current practice in separating functional and non-functional properties, non-functional properties are specified on a *per-service* basis (?, ?; Amir & Zeid, 2004; Ortiz & Hernändez, 2006; Lodderstedt, Basin, & Doser, 2002; Jřjens, 2002; Nakamura, Tatsubori, Imamura, & Ono, 2005; Soler, Villarroel, Trujillo, Medina, & Piattini, 2006; Vokäc, 2005; Baligand & Monfort, 2004; G. Wang, Chen, Wang, Fung, & Uczekaj, 2004). However, with a 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 scatter over multiple services simultaneously. For example, when a certain message encryption is consistently required throughout a purchasing business process, developers need to manually specify the encryption property for all services in the business process (e.g., retailers, suppliers, distributors and carriers) one by one. When a change occurs in the required encryption level, developers have to manually examine which services the change affects and carefully implement the change in the affected services. It is tedious, expensive and error-prone to consistently specify, maintain and validate non-functional properties on a per-service basis in a large-scale business process.

The second research issue this paper addresses is a lack of adequate methods to manage constraints among non-functional properties. In general, a series of constraints (e.g., dependency and mutual exclusion) exist among non-functional properties. For example, when messages are transmitted asynchronously between services, a timeout period should be specified as well to abort transmissions failures. Here, a dependency (or co-use) constraint exists between two non-functional properties: asynchronous message transmission and timeout. In order to maximize the reusability of services, non-functional constraints tend to be complicated and hard to maintain because their granularity becomes finer and their number grows. They are informally specified in natural languages in most of the current practice of separating functional and non-functional properties; it is tedious and error-prone for application developers to manually ensure that their applications satisfy required non-functional constraints (?, ?; Amir & Zeid, 2004; Ortiz & Hernändez, 2006; L. Wang & Lee, 2005; Nakamura et al., 2005). Few methods exist to explicitly specify non-functional constraints and consistently validate and enforce them in applications.

In order to address the above two research issues, this paper proposes (1) a programming language for a new strategy to separate functional and non-functional properties in business processes and (2) a graphical modeling method to specify, validate and enforce non-functional constraints in service-oriented applications. The proposed language, BALLAD[1], facilitates a *per-process* strategy to specify non-functional properties for business processes rather than services. BALLAD leverages the notion of *aspects* in aspect oriented programming/modeling (Kiczales et al., 1997; Elrad, Aldawud, & Bader, 2002) or *early aspects*, which are crosscutting concerns that

exist in early phases in application development process such as requirement analysis and business process design (Chitchyan et al., 2005). Each aspect is used to specify non-functional properties that crosscut multiple services in a business process. Given an aspect, a supporting tool, called Ark, identifies which services it is applied (or woven) to and automatically configures its corresponding non-functional properties to the identified services. This way, application developers do not need to manually specify and maintain non-functional properties for services one by one, thereby reducing the burdens/costs in development and maintenance of service-oriented applications.

The proposed modeling method, called FM-SNFPs, leverages the notion of feature modeling (Czarnecki & Eisenecker, 2000). Feature modeling is a simple yet powerful method to explicitly model a series of constraints among application's *features* (e.g., functionalities and configuration policies). By modeling a non-functional property as a feature, FM-SNFPs aids to graphically specify constraints among non-functional constraints and consistently validate the constraints in applications. Ark automatically enforces required non-functional constraints in applications by transforming their specification from business process models to application code (program code and deployment description) through intermediate models.

This paper overviews an end-to-end MDD framework that implements BALLAD and FM-SNFPs, and describes the design details of BALLAD and FM-SNFPs. It also demonstrates an application development with the proposed MDD framework. Empirical evaluation results show that BALLAD significantly reduces the burdens/costs to implement and maintain non-functional properties in service-oriented applications. BALLAD and FM-SNFPs are designed and implemented efficient and scalable.

The Proposed End-to-End MDD Framework

Figure 1 overviews the proposed MDD framework that implements BALLAD and FM-SNFPs. The framework consists of (1) BALLAD, (2) FM-SNFPs, (3) a Unified Modeling Language (UML) profile to specify non-functional properties in SOA, called UP-SNFPs (?, ?), and (4) a model transformation tool, called Ark. All artifacts in this framework are built and maintained with the metameta model (Ecore) in the Eclipse Modeling Framework (EMF[2]). The syntax of BALLAD is defined as a meta model on Ecore. FM-SNFPs is defined on the feature metamodel in fmp (Antkiewicz & Czarnecki, 2004)[3]. UP-SNFPs is defined as an extension (or profile) to the UML metamodel. Currently, Business Process Modeling Notation (BPMN) (*Business Process Modeling Notation (BPMN) 1.0*, 2004) is used as a language to graphically define business processes. BPMN models are defined with the BPMN metamodel in eBPMN[4]. Ark consists of two components, Ark.bpmn and Ark.uml, which transform BPMN models and UML models, respectively. Table 1 summarizes the artifacts and tools in the proposed MDD framework.

Figure 2 overviews the application development process using the proposed MDD framework. Application developers define a business process model in BPMN, an aspect in BALLAD, and a feature configuration(s) in FM-SNFPs. A feature configuration is an instance of FM-SNFPs, and it specifies a particular set of non-functional properties and their constraints that are used in a business process. A BALLAD aspect defines which feature configuration (i.e., a set of non-functional properties) is applied (or woven) to which model elements in a BPMN model. Ark.bpmn accepts a BPMN model, a BALLAD aspect and a feature configuration(s), and then transforms the BPMN model to a UML model represented with UP-SNFPs (an implementation-independent model in Figure 2) according to a given BALLAD aspect and feature configuration(s). During this transformation, Ark.bpmn automatically ensures that the generated

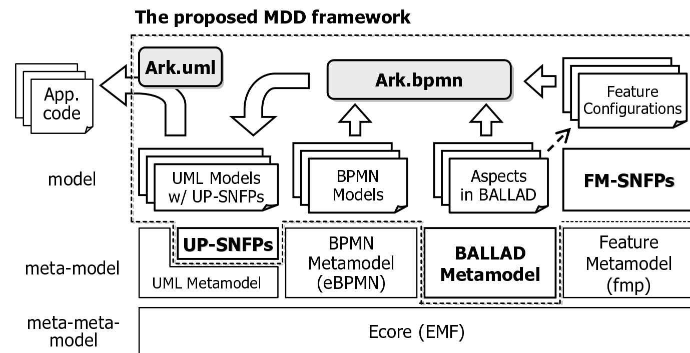

Figure 1. The Architecture of the Proposed MDD Framework

Table 1: Artifacts and Tools in the Proposed MDD Framework

Artifact/Tool	**Description**
BPMN	Business Process Modeling Language. A visual language to model business processes.
BALLAD	An aspect oriented language to define which non-functional properties are applied (or woven) to which model elements in a BPMN model.
FM-SNFPs	A feature model that defines a series of non-functional properties and constraints among them.
Feature configuration	An instance of FM-SNFPs. Each feature configuration specifies a particular set of non-functional properties and their constraints that are used in a business process.
UP-SNFPs	A UML profile that represents non-functional properties in UML.
Ark.bpmn	A model transformer that transforms a BPMN model into a UML model represented with UP-SNFPs according to a given BALLAD aspect and feature configuration(s).
Ark.uml	A model transformer that transforms a UML model represented with UP-SNFPs into application code (program code and deployment descriptor).

UML model satisfies a set of constraints among non-functional properties defined in FM-SNFPs. Once an implementation-independent UML model is generated, application developers can add methods (or behaviors) to individual UML classes and map it to an implementation-specific UML model by specifying various parameters for implementation technologies such as transport protocols and Enterprise Service Buses (ESBs). Ark.uml transforms an implementation specific UML model into a skeleton of application code (program code and deployment descriptors).

Using BALLAD, non-functional properties can be specified for business processes in an implementation independent manner. They are portable and reusable across different implementation technologies. Throughout a chain of model transformations, they are separated from functional properties and gradually mapped from abstract implementation-independent models to application code through concrete implementation-specific models.

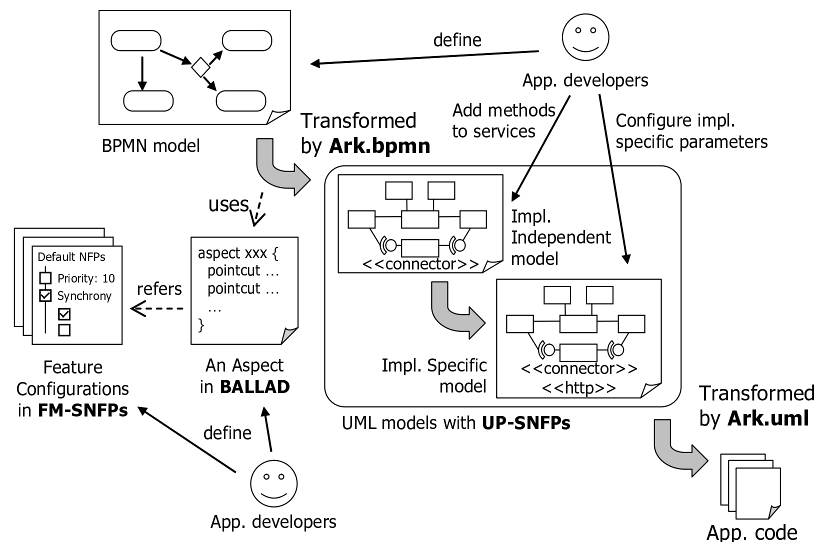

Figure 2. Application Development Process with the Proposed MDD Framework

Background: Business Process Modeling Notation (BPMN)

This section briefly overviews BPMN. BPMN is a visual modeling language to define business processes. Figure 3 shows an electronic voting process in a certain online community. It involves four entities: `Secretary`, `Moderator`, `Assistant Moderator` and `Voting Member`[5]. A secretary examines, on every Friday, whether there exist any issues to discuss and decide. If there is not, the secretary sends an issue list to a moderator. The moderator initiates a discussion among voting members and waits for their votes for a week. An assistant moderator receives and reviews collected votes to examine whether if an issue in question is settled. If not, the issue is discussed for a revote.

A BPMN model consists of *pool*s, *task*s and *sequence/message flow*s. A pool, represented by a rectangle, denotes a participant in a business process; for example, `Secretary` in Figure 3. A task, represented as a rounded-corner rectangle, denotes a task performed by a participant; for example, `Receive Issue List` in `Secretary`. 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.

In addition to tasks, pools can contain *gateway*s and *event*s. A gateway, represented as a diamond shape, controls the divergence (forking) and convergence (merging) of sequence flows. For a divergence of sequence flows, a gateway can have the default sequence flow, represented as a sequence flow with a slash mark, which is chosen if other sequence flows are not selected. In Figure 3, a `Secretary` branches a flow depending on the existence of issues to discuss and vote. If issues exist, it performs `Send Issue List`; otherwise, it selects the default sequence flow and reaches `Finish`.

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 icon inside, denotes a reception of a message from a participant. In Figure 3, a `Moderator` triggers its process when it receives an `Issue List`

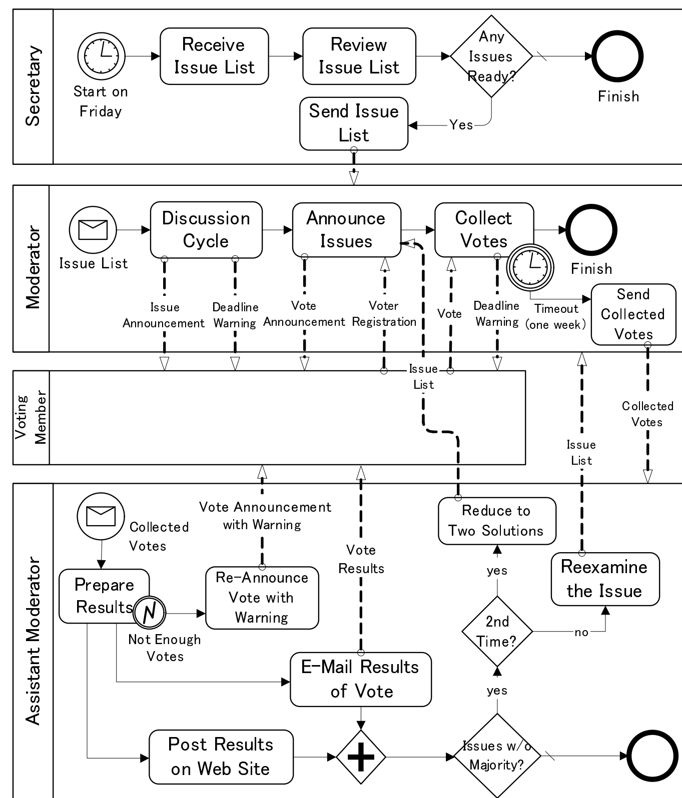

Figure 3. A Voting Process Model in BPMN

message. A timer, represented as a circle with a clock, denotes a specific time or interval. In Figure 3, a `Moderator` performs `Send Collected Votes` one week after it performs `Collect Votes`. An error, represented as a circle with a lightning inside, denotes a specific error condition. In Figure 3, an `Assistant Moderator` performs `Re-Announce Vote with Warning` when it does not receive enough votes in `Prepare Result`.

BALLAD: 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 (Kiczales et al., 1997; Elrad et al., 2002). Then, supporting tools (often called *aspect weavers*) weave aspects into the other parts of an application to complete it. An aspect consists of *advice*s and *pointcut*s. An advice defines or implements a concern that appears (or crosscuts) many places in an application. A pointcut specifies the places where advices appear.

As described in the Introduction section, non-functional properties are crosscutting concerns. Thus, BALLAD is designed to modularize a set of non-functional properties used in a business process as an aspect. In BALLAD, an advice is defined as a feature configuration, in FM-SNFPs, which specifies a set of non-functional properties. A pointcut specifies the places to which advices (i.e., non-functional properties) are woven in a BPMN model. Advices are visually defined, while pointcuts are defined in a textual form. Ark.bpmn serves as an aspect weaver for BALLAD; it

weaves non-functional properties into a BPMN model according to the definition of pointcuts. (See also Figures 1 and 2.) 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.

Listing 1 shows an example aspect in BALLAD. An aspect is defined with the keyword `aspect`, followed by its name. An aspect can define an arbitrary number of pointcuts and references to advices. A pointcut is defined with the keyword `pointcut`, followed by its name. Advices are defined separately from an aspect, and the aspect references them with their names.

In Listing 1, the pointcut `discussion` specifies paths between two model elements (task or message flow) in a BPMN model by using the `within` join point. (A join point is a place where advices can be woven, and a pointcut is a set of join points which actually used.) The `within` join point takes two names of model elements in regular expression as parameters. (`::` represents a separator between model elements such as Pools and Tasks.) A set of model elements matched the first parameter and the second parameter are interpreted as starting points and ending points of paths respectively. For example, the pointcut `discussion` specifies paths between `Moderator::Discussion.*` and `Moderator::Finish`. As Figure 3 shows, these two parameters match the tasks `Discussion Cycle` and `Finish` in `Moderator` respectively, and there are many paths between them. For example, the shortest path starts from `Moderator::Discussion Cycle`, passes through `Moderator::Announce Issues` and `Moderator::Collect Votes`, and reaches `Moderator::Finish`. Several paths go through the Timer event attached to `Moderator::Collect Votes`, pass Tasks in `Assistant Moderator` and return to `Moderator`. This way, Ark.bpmn automatically finds a set of model elements that involved in (sub)processes according to pointcuts.

Listing 1 An Example Aspect in BALLAD

```
aspect NFPsForVoting{
 // pointcuts
 pointcut discussion: within("Moderator::Discussion.*", "Moderator::Finish");
 // references to advices
 discussion: DiscussionNFPs;
}
```

Ark.bpmn weaves advices into model elements in a BPMN model according to given references to advices, each of which consists of a pointcut name and advice name (`discussion` and `DiscussionNFPs` in Listing 1). When an aspect references multiple advices, Ark.bpmn weaves them to BPMN model elements following the order of their occurrence.

Listing 2 shows the syntax of BALLAD. As shown in the figure, BALLAD supports other join points than `within`: `target`, `source`, `flow`, `trigger`, `depth` and `default` (Table 2). Listing 3 shows an example aspect that uses these extra join points. In this example, `DeliveryAssurance` references an advice that represents a set of non-functional properties for assured/reliable message delivery. `HighlevelSecurity` references an advice regarding message encryption and access control enforcement.

`target` and `source` are simplified representations of `within( ".*", "PoolName::.*" )` and `within( "PoolName::.*", ".*" )`, where `PoolName` serves as a parameter of `target` and `source`. They return all paths that arrive at and depart from certain pools, respectively. For example, in Listing 3, the pointcut `fromVotingMember` selects all paths departing from `Voting`

Listing 2 The Syntax of BALLAD

```
<aspect> ::= 'aspect' <aspectName> '{' (<pointcut>)+ (<adviceref>)+ '}'
<pointcut> ::= 'pointcut' <pointcutName> ':' <joinpoint>* ';'
<adviceref> ::= <pointcutName> ':' <adviceName> (',' <adviceName>)* ';'

<joinpoint> ::= <mandatoryjp> ('&&' <mandatoryjp> | <optionaljp>)*
<mandatoryjp> ::= <within> | <target> | <source> | <flow> | <trigger>
<optionaljp> ::= <depth> | <default>
<within> ::= 'within' '(' <elementName> ',' <elementName> ')'
<target> ::= 'target' '(' <poolName> ')'
<source> ::= 'source' '(' <poolName> ')'
<flow> ::= 'flow' '(' <flowName> ')'
<trigger> ::= 'trigger' '(' <eventType> ')'
<eventType> ::= 'MESSAGE' | 'TIMER' | 'ERROR' | 'CANCEL' | 'COMPENSATION' | 'RULE'
<depth> ::= 'depth' '(' <integer> ')'
<default> ::= 'default' '(' ')'

<aspectName> ::= <identifier>
<pointcutName> ::= <identifier>
<adviceName> ::= <identifier>
<elementName> ::= '"' <regex> ('::' <regex>)* '"'
<poolName> ::= '"' <regex> '"'
<flowName> ::= '"' <regex> '"'
<regex> ::= // regular expression
```

Listing 3 An Example Aspect in BALLAD

```
aspect NFPAspect{
 // pointcuts
 pointcut wholeProcess: within("Secretary::Start.*", "Secretary::Finish");
 pointcut fromVotingMember: source("Voting Member");
 pointcut vote: flow("Vote");
 pointcut error: trigger(ERROR);
 pointcut secretary:
      within("Secretary::Start.*", "Secretary::Finish") && depth(1);
 pointcut defaultProcess:
      within("Secretary::Start.*", "Secretary::Finish") && default();
 pointcut afterVoting: trigger(TIMER) && target("Assistant Moderator");

 // references to advices
 wholeProcess: DefaultSecurity, NoDeliveryAssurance;
 fromVotingMember: MessgeEncryption;
 vote: HighlevelSecurity;
 error: DeliveryAssurance;
 secretary: NoSecurity;
 defaultProcess: DeliveryAssurance;
 afterVoting: NoDeliveryAssurance;
}
```

Table 2: Join Points in BALLAD

Join Point	Description
within	Returns all paths between two model elements (tasks or message flows).
target	Returns all paths arriving at certain pools.
source	Returns all paths departing from certain pools.
flow	Specifies certain message flows.
trigger	Returns all paths departing from a certain type of event.
depth	Limits the number of unique pools in a path.
default	Follows only the default sequence flows at gateways.

`Member` (See also Figure 3.) `target` and `source` aid in specifying non-functional properties that certain pools require for their incoming/outgoing message flows. In Listing 3, a set of message encryption properties is specified for all outgoing message flows from `Voting Member`.

`flow` is a simplified representation of `within( "FlowName", "FlowName" )`, where `FlowName` serves as a parameter of `flow`. It directly specifies particular message flows. For example, in Listing 3, the pointcut `vote` selects the message flow `Vote` in Figure 3. `flow` aids in defining non-functional properties that specific message flows require. In Listing 3, the `Vote` message flow requires higher security level than the other message flows do.

`trigger` returns all paths that start from a certain type of events (i.e., message, timer, rule, error, cancel or compensation). For example, in Listing 3, the pointcut `error` selects paths starting from error events in order to specify non-functional properties for error handling with message delivery assurance enabled.

`depth` is used with other join points. It limits the maximum number of unique pools to be included in a path(s) selected by other join points. For example, a selected path can contain up to three unique pools when a pointcut declares `depth(2)`. In Listing 3, the pointcut `secretary` uses `depth`. Although its `within` joint point returns all paths in Figure 3, it returns the paths between `Secretary` and `Moderator` because of `depth(1)`.

`default` is used with other join points. It selects the paths containing default sequence flows at gateways. For example, in Listing 3, the pointcut `defaultProcess` selects a default sequence flow at a gateway.

Join points can be used together. For example, Listing 3 defines the pointcut `afterVoting`, which uses `trigger` and `target` to select the paths starting from timer events and ending with the tasks in `Assistant Moderator`. When a pointcut uses multiple join points, Ark.bpmn returns the intersection of paths selected by them. For example, the pointcut `afterVoting` returns a set of paths contained in both `trigger(TIMER)` and `target("Assistant Moderator")`.

FM-SNFPs: A Feature Model for Non-Functional Properties in SOA

A feature model describes a set of features and constraints among them through a hierarchical (or tree) structure. Application developers create an instance of a feature model, i.e., a feature configuration, by selecting features for their applications on a supporting tool. By not allowing users to create feature configurations that violate constraints, a supporting tool assures that feature configurations satisfy constraints among features defined in a feature model.

FM-SNFPs is a feature model that defines a set of non-functional properties in SOA and

constraints among them, and aspects in BALLAD refer feature configurations of FM-SNFPs as their advices. FM-SNFPs covers the following four areas of non-functional properties.

- *Message Transmission Semantics*: messaging synchrony, message delivery assurance, message queuing, multicast, manycast, anycast, message routing, message prioritization, messaging timeout, message logging, and message retention.
- *Security Semantics*: transport-level encryption, message-level encryption (entire/partial message encryption), message signature, message access control, service access control, and secure conversation.
- *Service Deployment Semantics*: service redundancy.
- *Transport Protocols*: properties in certain transport protocols

Implementation Independent Non-Functional Properties in FM-SNFPs

Figure 4 shows the feature model in FM-SNFPs. There are several constraints and relationships among non-functional properties (Table 3). White and black circle icons indicate optional and mandatory non-functional properties, respectively. For example, `Message Priority`, `In Order Transmission` (i.e., specifies whether the order of messages that a message destination receives is same as the order of messages that a message source sends out) and `Message Integrity` (i.e., checks whether messages are altered during their transmission) are optional. `Retransmission` is also optimal, however `Number` (the maximum number of retransmissions) is mandatory. Therefore, `Number` must be selected when `Retransmission` is selected. A feature may have its type. For example, the type of the `Message Priority` feature is `Integer`. When a typed feature is selected in a feature configuration, its value (e.g., integer value) must be specified at the same time.

A fork icon with a white sector denotes an *exclusive-OR* relationship among non-functional properties. In Figure 4, one of `Sync`, `Async` or `Oneway` must be selected for `Synchrony`. A fork icon with a black sector denotes an *OR* relationship among non-functional properties. Cardinality indicates the number of subfeatures to be selected. `Delivery Assurance` feature has two subfeatures, and one or two of them should be selected when `Delivery Assurance` is selected. When cardinality is omitted, the default cardinality, i.e., one to the number of subfeatures, will be used (Czarnecki, Helsen, & Eisenecker, 2005). (`Delivery Assurance` shows the default cardinality explicitly.) `At Least Once` means that a connector retries delivering a message until its destination receives the message. However, the message may be delivered to its destination more than once. `At Most Once` 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. When both are selected, a message is delivered to its destination exactly once.

`Logging` has three subfeatures: `Message Transmission`, `Message Routing` and `Message Revision`. The features make application logs revision to auditable for the third party organizations in the future. `Message Transmission` and `Message Routing` specify whether messaging middleware and messages to retain logs on message transmissions, respectively. When `Message Transmission` is selected, messaging middleware logs (1) which messages are transmitted, (2) message source and destination, and (3) when the messages are transmitted. When `Message Routing` is selected, messages records (1) message source and destination, and (2) when the messages are transmitted (e.g., in their header). `Message Revision` specifies whether to retain message's revision history. 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

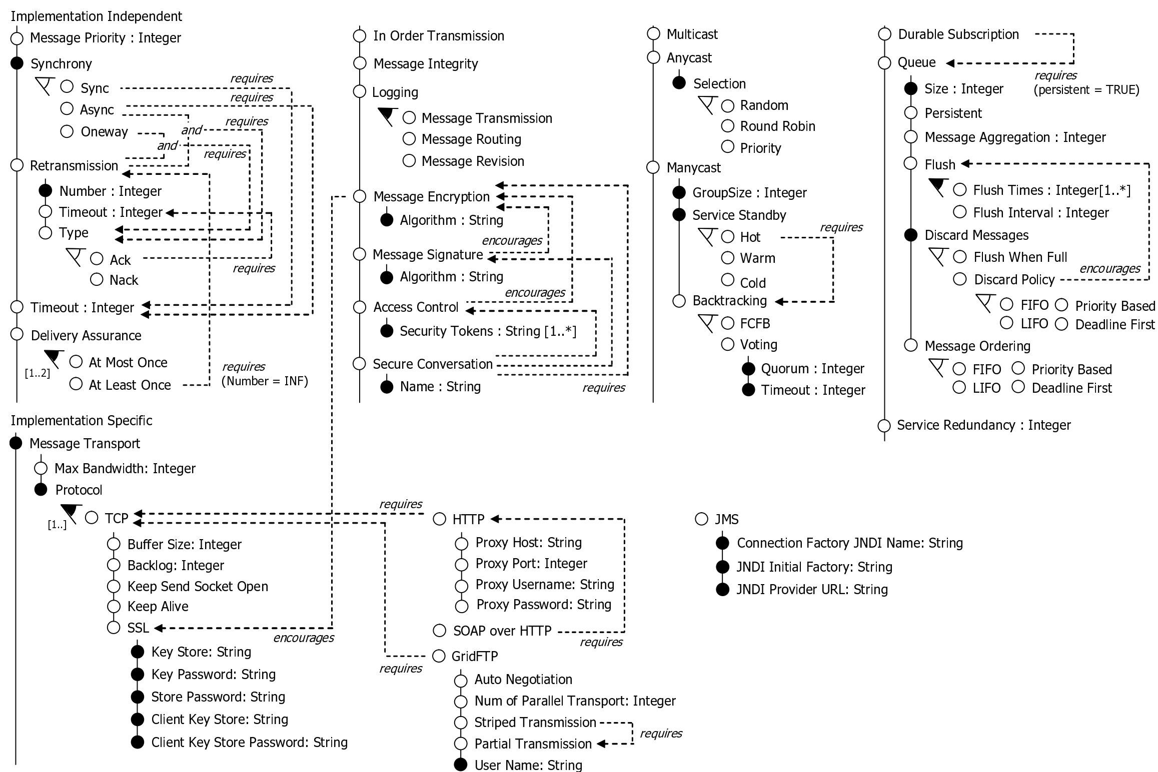

Figure 4. Definition of FM-SNFPs

revised them.

In addition to the hierarchical structure of features, three relationships among features are supported: `requires`, `encourages` and `discourages` (Table 3). A `requires` relationship indicates a dependency among non-functional properties. For example, when `Async` and `Retransmission` are selected, `Type` of `Retransmission` (Ack-base or Nack-base) must be selected too. If the type of retransmission is Ack-base, `Timeout` must be selected too for configuring a timeout period of Ack messages. `Secure Conversation` requires `Message Encryption`, `Message Signature` and `Access Control` that used to establish secure connections shared with services.

`encourages` and `discourages` relationships are newly introduced in this research project. `encourages` is a relationship which is similar but weaker than `requires`. It has no mandatory power, but endorses to use the referred features at the same time. For example, in Figure 4, the `Access Control` feature *encourages* to select the `Message Encryption` feature at the same time. Selecting `Message Encryption` with `Access Control` makes systems much secure because transmitting security tokens via unsecured connections makes systems vulnerable, but in-house services may not require message encryption but need access control for the purpose of audit. `discourages` relationship works in direct contrast to `encourages` relationship, and it discourages to use referred features. By showing messages a supporting tool encourages or discourages application developers to select certain non-functional properties. This way, `encourages` and `discourages` relationships facilitate the decision-making process in designing

Table 3: Constraints/Relationships of Features

Constraint/Relationship	Description
mandatory feature	A feature that must be selected.
optional feature	A feature that optionally be selected.
exclusive-OR	A mutual exclusion constraint among features. It enforces to select one of subfeatures.
OR	An inclusive relationship among features. It enforces to select one or more subfeatures.
requires	A dependency among features. It enforces to use the referred features at the same time.
encourages	A weak dependency among features. It endorses to use the referred features at the same time.
discourages	A weak mutual exclusion constraint among features. It discourages to use the referred features at the same time.

applications.

Implementation Specific Non-Functional Properties in FM-SNFPs

In addition to non-functional properties independent from implementation technologies, FM-SNFPs defines non-functional properties specific to certain implementation technologies (Figure 4). `Protocol` has five subfeatures: `TCP`, `HTTP`, `SOAP over HTTP`, `GridFTP` and `JMS`.

`TCP` represents TCP connections, and it has five subfeatures. `Buffer Size` specifies the buffer size (byte) used to read and write data. `Backlog` specifies the maximum queue length for incoming connections. `Keep Send Socket Open` specifies whether to reuse a single socket for multiple dispatches and keep it open until an application explicitly close it. `Keep Alive` specifies whether to send a packet via an unused open socket to a remote server every certain period. `SSL` is for configuring SSL (TSL) connection. `Key Store` and `Client Key Store` specify the location (a file path) of a server and client keystore used to create a secure socket respectively. `Store Password` and `Client Key Store Password` specifies the password used to access server and client key stores respectively. `Key Password` specifies a password used to check the integrity of keys. When a connector uses SSL, these five subfeatures must be selected and configured.

`HTTP` represents HTTP connections, and it has four subfeatures. When a connection uses a proxy host, `proxyHost` and `proxyPort` must be selected. Also, if a proxy host requires a user name and password, `proxyUsername` and `proxyPassword` must be selected. Since `HTTP` requires `TCP`, `TCP` must be selected with `HTTP` at the same time. `SOAP over HTTP` represents connections via SOAP over HTTP. Since it requires both `TCP` and `HTTP`, they must be selected with `SOAP over HTTP` at the same time.

`Grid FTP` represents connections via GridFTP[6], and it has five subfeatures. `Auto Negotiation` enables an auto TCP buffer tuning that GridFTP supports. GridFTP allows for using multiple streams (connections) to transmit data, and `Num of Parallel Streams` specified the number of connections to use. GridFTP allows for a sink node to receive data from multiple source nodes at a time, and `Stripe dStreams` enables it. GridFTP allows for transmitting a fragment of data, and `Partial Streams` enables it. Also, `User Name` specifies the username to be used in a

GridFTP server.

`JMS` represents connections via Java Messaging Service, and it has three subfeatures. `JNDI Initial Factory` specifies a class used to access to a Java Naming and Directory Interface (JNDI) server. `Jnid Provider URL` specifies a URL to access a JNDI server. `Connection Factory JNDI Name` specifies a name of a JNDI context to obtain a connection to JMS.

Feature Configurations of FM-SNFPs

Figure 5 shows a feature configuration of FM-SNFPs. In this feature configuration, several features (e.g., `Async` subfeature and `Retransmission` feature) are selected. A supporting tool resolves the constraints between (sub)features. For example, the `Sync` and `Oneway` subfeatures are automatically deselected when the `Async` subfeature is selected. Also, it reports missing properties (e.g., an alert is shown when the value of the `Timeout` is not configured even though the feature is selected.) and existence of encouraged features (e.g., a message is shown to encourage to select `Message Encryption` when `Access Control` is selected.) to developers.

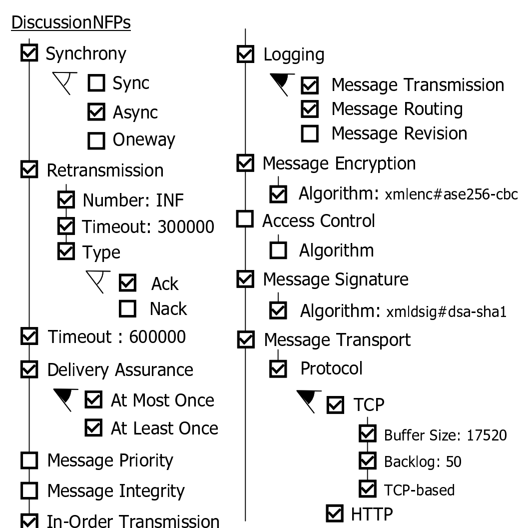

Figure 5. An Example of a Feature Configuration

Ark.bpmn weaves feature configurations into a BPMN model in order of the references to advices. For example, Listing 3 weaves non-functional properties in `DefaultSecurity` and `NoDeliveryAssurance` into the pointcut `wholeProcess`. Since it is the first advice referred in an aspect, Ark.bpmn weaves the two sets of non-functional properties into a BPMN model first. Then, `MessageEncryption` is woven into `fromVotingMember`. 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, BALLAD 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., pools and message flows in a certain sub-process) by only changing pointcuts. Also, it is easy to specialize certain non-functional properties in a sub-process by

overwriting non-functional properties by introducing new pointcuts. This separation makes easy to configure applications in typical situations (e.g., services hosted in-house, or accessed via the Internet) by reusing existing feature configurations.

UP-SNFPs: A UML profile for Non-Functional Properties in SOA

By weaving an aspects into a BPMN model, a BPMN model is transformed into a UML model with non-functional properties (Figure 2). Non-functional properties in a UML model is described through UP-SNFPs (?, ?). UP-SNFPs is a UML profile to visually specify non-functional properties in UML's class and composite structure diagrams. It is designed around two major concepts: *service*s and *connector*s between services. Each connector defines how services are connected with each other and how messages are exchanged through it. UP-SNFPs can be used to define implementation independent and implementation specific models with non-functional properties.

Implementation Independent Models with UP-SNFPs

Figure 6 shows an example model defined with UP-SNFPs. It illustrates a voting management application, which corresponds to a sub-process of the voting process in Figure 3. In this example, three services (`Moderator`, `AssistantModerator` and `VotingMember`) exchange messages. Each service is represented by a class stereotyped with ≪`service`≫.

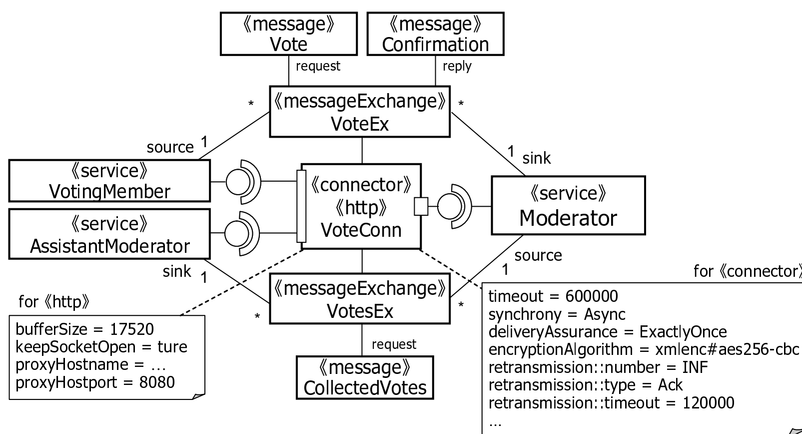

Figure 6. An Example of a UML Model with UP-SNFPs

The services in Figure 6 exchange three types of messages (`Vote`, `Confirmation` and `CollectedVotes`), 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 `Vote` (request) and `Confirmation` (reply) messages is represented by `VoteExchange` in Figure 6.

≪`connector`≫ represents a connection that transmits messages between services. In Figure 6, messages are delivered through the connector `VoteConn`. 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 icon) and a required interface (represented as a socket icon). Services use the provided and required interfaces to send and

receive messages, respectively. Each connector can have multiple tagged-values to specify a set of message transmission and processing semantics. In Figure 6, the connector `VoteConn` specifies the timeout of message transmissions (600,000 milliseconds), synchrony of message transmissions (asynchronous), assurance level of message delivery (exactly once), parameters of message retransmission (ack base and its timeout is 120,000 milliseconds), and an encryption algorithm for messages (Advanced Encryption Standard).

Implementation Specific Models with UP-SNFPs

In addition to stereotypes designed for implementation independent models, UP-SNFPs supports several stereotypes specific to implementation technologies such as transport protocols and middleware. They inherit the `Connector` stereotype (Figure 7), and used to annotate connectors to specify to which implementation technologies the connectors are mapped. For example, services in Figure 6 exchange messages via HTTP since the connector `VoteConn` is stereotyped with ≪`http`≫.

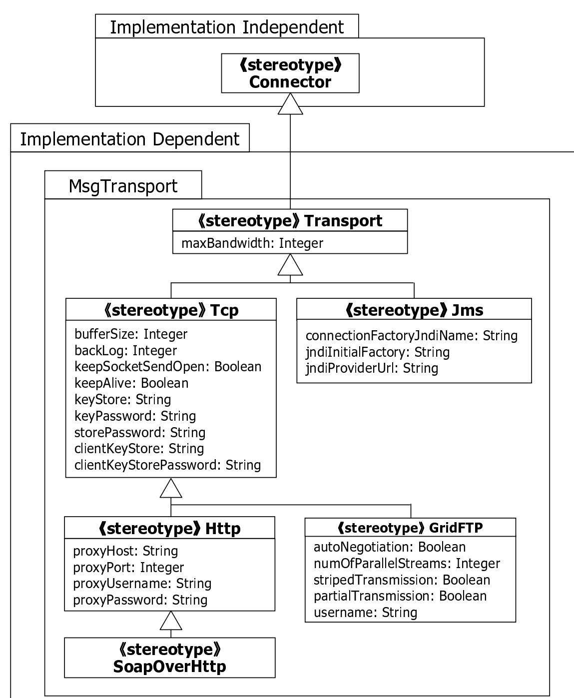

Figure 7. Stereotypes for Implementation Technologies

When a connector in an input UML model has a stereotype for a certain implementation technology, Ark.uml generates application code that uses the implementation technology. When a connector does not have stereotypes for certain implementation technologies, Ark.uml generates code but it is not bound with any implementation technology and developers are required to configure the generated code by hand to use certain implementation technologies.

Application Development with Ark

Figure 2 shows the application development process with the proposed aspect oriented language. Ark.bpmn takes a BPMN model and an aspect in BALLAD, 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.

Defining an Aspect in BALLAD and FM-SNFPs

For defining aspects, Ark provides an editor for BALLAD and a modeling tool for FM-SNFPs running on Eclipse. Figure 8 illustrates the editor for BALLAD. As it 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 openArchitectureware[7] (oAW). oAW allows developers to define the syntax of user-defined languages in EMF (BALLAD metamodel in Figure 1), and based on the syntax in EMF oAW generates editors for the languages (Figure 8).

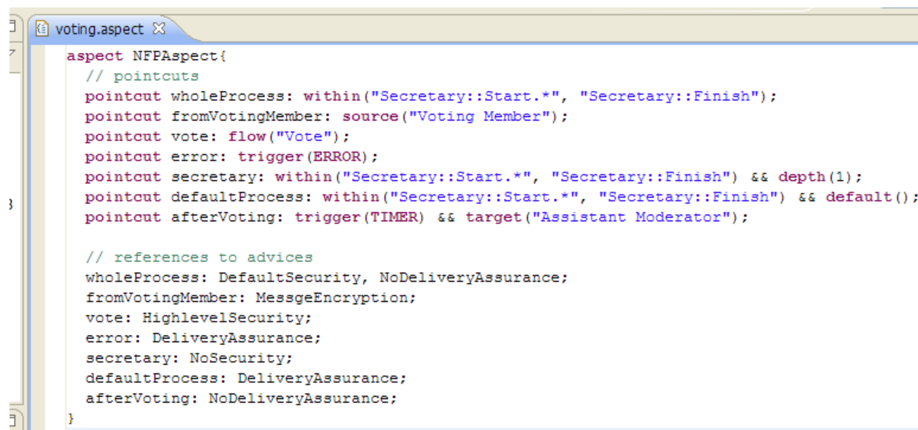

Figure 8. A Screenshot of an Editor for BALLAD

FM-SNFPs is defined on fmp (Antkiewicz & Czarnecki, 2004), which is a feature modeling tool implemented on EMF. This research project extends fmp to support `encourages` and `discourages` relationships. Figure 9 shows a feature configuration in a feature modeling tool. The editor does not allow users to create feature configurations that violate constraints defined in a feature model. For example users can select only one of features in mutual exclusion at a time. Also, the editor notifies users of missing information, `encourages` relationships and `discourage` relationships. In Figure 9, messages shown in the bottom notify the existence of a missing value in the `Timeout` feature and a `encourages` relationship.

Transformation from BPMN to UML

Ark.bpmn performs a model transformation from BPMN to UML in two steps: (1) transforming a BPMN model into a UML model without non-functional properties, and (2) configuring non-functional properties on the generated UML model based on the definition of an aspect in BALLAD.

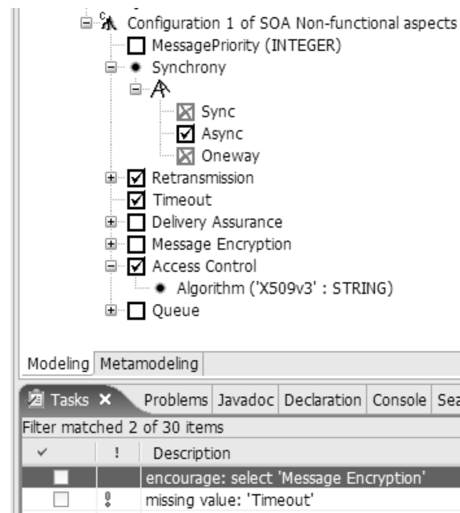

Figure 9. A Screenshot of a Feature Modeling Tool

The first step simply transforms a BPMN model into a UML model by following the transformation rules shown in Table 4. A generated UML model has several stereotypes defined in UP-SNFPs (e.g., ≪service≫ and ≪connector≫), but does not have any non-functional properties yet. Figure 10) is a fragment of a UML model transformed from the BPMN model in Figure 3. (The UML model contains model elements corresponding to the `Moderator` and `Voting Member` pools, and the `Issue Announcement` and `Vote` message flows in Figure 3.) Ark.bpmn transforms a pool in a BPMN model into a class stereotyped with ≪service≫. (e.g., the `Moderator` pool is mapped into the `Moderator` class.) Each task in a pool is transformed into a method in a class. (e.g., the `Discussion Cycle` task is mapped into the `discussionCycle` method.) Also, a message flow between 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 respectively. (Since a message flow in BPMN represents an oneway message, only a request message is generated in a UML model.)

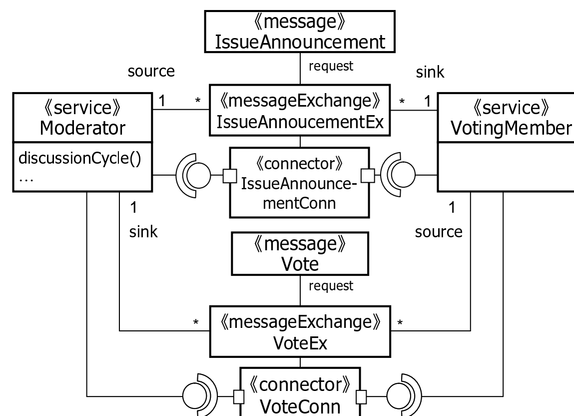

Figure 10. A Fragment of a Generated UML model

Table 4: Mapping Rules from BPMN to UML

Model Element in BPMN	Model Element in UML
A pool	A class with ≪service≫ with the same name. (whitespace characters are removed.)
A task	A method with the same name (whitespace characters are removed and the first character is decaptalized.)
A message flow	A class with ≪connector≫ with the name of *flowname*Conn A class with ≪messageExchange≫ with the name of *flowname*Ex A class with ≪message≫ with the name of *flowname* (where *flowname* refers the name of a message flow)
An outgoing message flow from a pool	An association role 'source' against a message exchange
An incoming message flow to a pool	An association role 'sink' against a message exchange

Configuring Non-Functional Properties in a UML model

Ark.bpmn parses an aspect in BALLAD and finds which feature configurations are applied to which model elements in a given BPMN model. When a feature configuration is applied to certain tasks and/or message flows in a BPMN model, the feature configuration is applied to services in a UML model that have methods corresponding to the tasks and/or connectors corresponding to the message flows. For example, the feature configuration `DiscussionNFPs` is applied to the pointcut `moderator` in Listing 1. Since the pointcut `moderator` returns paths that contain several tasks in `Moderator` and the `Voting Member` pool in a BPMN model (Figure 3), non-functional properties in the feature configuration `DiscussionNFPs` is applied to `Moderator` and `Voting Member` services in a generated UML model (Figure 10). Also, since the paths contains several message flows between `Moderator` and `Voting Member`, the feature configuration `DiscussionNFPs` is applied to corresponding connectors in a generated UML model as well.

Then, Ark.bpmn saves the result in an XML file. Listing 4 is a fragment of the result obtained from `NFPAspect` in Listing 3. It lists pairs of the name of a feature configuration and a set of model elements where the feature configuration applies. As illustrated, the result contains a set of `featureconfiguration` tags. Each of them specifies which feature configuration is applied to which model elements using its `name` attribute and `model` tags respectively. A `model` tag has the attribute `pattern` specifying the name of a model element.

Ark.bpmn takes this XML file and configures tagged-values of model elements in a generated UML model. Listing 5 shows the process. Ark.bpmn traverses a generated UML and checks each model element (`e` in Listing 5) whether any feature configurations (`featureConf` in Listing 5) are applied. If yes, Ark.bpmn configures the model element's tagged-values according to feature configurations. Figure 11 is a UML model that Ark.bpmn generates by weaving the feature configuration in Figure 5 into the UML model in Figure 10.

Listing 4 A Result Saved in XML

```
<weaving>
 <featureconfiguration name="DefaultSecurity">
  <model pattern="Securetary"/>
  <model pattern="Moderator"/>
  ...
  <model pattern="IssueAnnoucement"/>
  ...
 </featureconfiguration>
 <featureconfiguration name="NoDeliveryAssurance">
  <model pattern="Securetary"/>
  <model pattern="Moderator"/>
  ...
  <model pattern="IssueAnnoucement"/>
  ...
 </featureconfiguration>
 <featureconfiguration name="MessageEncryption">
  <model pattern="Moderator"/>
  <model pattern="VotingMember"/>
  <model pattern="Votes"/>
  ...
 </featureconfiguration>
 ...
</weaving>
```

Listing 5 Pseudo Code for Weaving Process

```
weaving( UMLElement e, FeatureConfiguration[] featureConfs ){
 foreach featureConf in featureConfs{
  if (featureConf is not applied to e)
   return;

  if (e is stereotyped with <<messageExchange>>)
   if (featureConf has a 'Message Priority' value)
    configure connector's 'priority' tagged-value
   if (featureConf has a 'Synchrony' value)
    configure connector's 'synchrony' tagged-value
   ...
  else if (e is stereotyped with <<service>>)
   if (featureConf has 'AccessControl')
    replace <<service>> with <<accessControlledService>>
    configure service's 'securityTokens' tagged-vale
   ...
 }
}
```

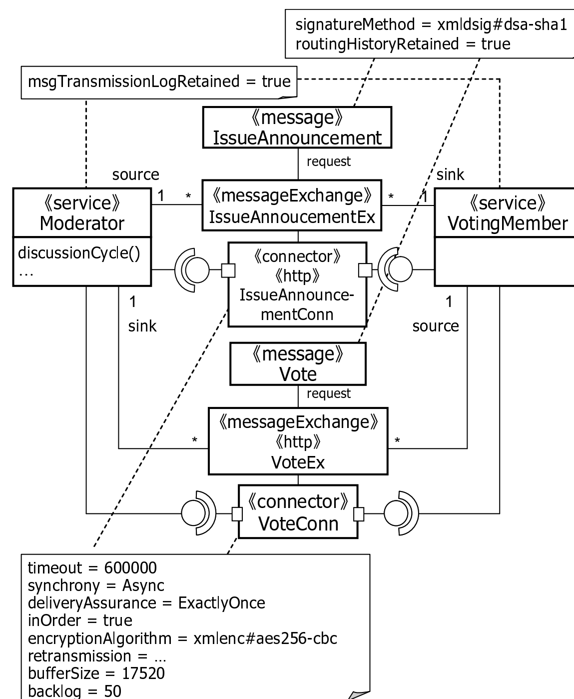

Figure 11. A UML model with Non-Functional Properties

Code Generation

Once Ark.bpmn completes its model transformation, Ark.uml transforms a UML model with UP-SNFPs to a skeleton of application code (program code and deployment descriptors). (See Figures 1 and 2). Currently, Ark.uml implements a transformation mapping for two major ESBs, Mule ESB[8] and ServiceMix ESB[9], and GridFTP.

When Mule ESB is selected as middleware to operate applications, Ark.uml transforms a UML class stereotyped with ≪`message`≫ to a Java class that has the same class name. The Java class implements the interface `Serializable`. This is required to implement messages exchanged in Mule ESB. A UML class stereotyped with ≪`service`≫ is transformed to a Java class that has the same class name and the same methods. Also, Ark.uml inserts several methods to the Java class depending on whether its association role is source/sink against a message exchange.

Although UML classes stereotyped with ≪`messageExchange`≫ and ≪`connector`≫ are not transformed to particular Java classes, Ark.uml generates a corresponding deployment descriptor to configure connectors between services according to the message transmission/processing semantics specified in a UML model. Listing 6 shows a fragment of a deployment descriptor generated from the UML model in Figure 11. `<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, its value is `http://....` `<mule-descriptor>` specifies the name (`name`) and implementation (`implementation`) of a service, `<inbound-router>` specifies the URL of a service by referencing an end point, and `<connector>` specifies a transport protocol to deliver messages. Since the `VoteConn` connector in Figure 11 is stereotyped with ≪`http`≫, the generated deployment descriptor is configured to use `org.mule.providers.http.HttpConnector`,

which is provided by Mule ESB, to access services via HTTP. Also, properties of a transport protocol are specified in `<property>`. See (?, ?) for full discussion on a transformation mapping for implementation independent stereotypes and tagged-values in UP-SNFPs.

Listing 6 An Example of a Deployment Descriptor

```
<mule-configuration>
 <endpoint-identifiers>
  <endpoint-identifier name="Moderator_in_VoteEx" value="http://..."/>
 </endpoint-identifiers>
 <model>
  <mule-descriptor name="ModeratorService" implementation="Moderator">
   <inbound-router>
    <endpoint address="Moderator_in_VoteEx" connector="VoteConn"/>
   </inbound-router>
  </mule-descriptor>
  <mule-descriptor name="VotingMemberService" implementation="VotingMember"/>
 </model>
 <connector name="VoteConn" className="org.mule.providers.http.HttpConnector">
  <properties>
   <property name="bufferSize" value="17520"/>
   <property name="backlog" value="50"/>
  </properties>
 </connector>
</mule-configuration>
```

Extensibility of the Proposed MDD Framework

Because of the huge diversity of non-functional properties, applications may require new non-functional properties and constraints that are not supported in the current proposed MDD framework. In order to introduce new non-functional properties and constraints, application developers are required to extend FM-SNFPs, UP-SNFPs and transformation rules. Since BALLAD does not depend on the definition of non-functional properties, any extensions have no affects on BALLAD.

Since the editor for FM-SNFPs (Figure 9) allows for extending the FM-SNFPs's model, application developers can easily add new non-functional properties and constraints to FM-SNFPs. UP-SNFPs is built on the UML standard metamodel with the standard extension mechanism, and application developers can add stereotypes and tagged-values representing new non-functional properties. In addition to the extension to FM-SNFPs and UP-SNFPs, extending transformation rules from FM-SNFPs to UP-SNFPs and rules from UP-SNFPs to application code completes the extension of the proposed MDD framework.

Empirical Evaluation

This section empirically evaluates BALLAD, FM-SNFPs and Ark.bpmn. The execution overhead of Ark.bpmn was measured with a Sun Java SE 6.0 VM running on a Windows XP PC with an AMD Sempron 3.0 Ghz CPU and 1024 MB memory space.

Separation of Functional and Non-functional Properties

In order to evaluate how BALLAD separates functional and non-functional properties effectively, this paper uses two metrics: Concern Diffusion over Components (CDC) (Sant'Anna, Garcia, Chavez, Lucena, & Staa, 2003) and Degree of Scattering (DOS) (Eaddy, Aho, & Murphy, 2007). Both are measured for a BPMN model with a BALLAD aspect and a UML model generated by Ark.bpmn. CDC counts the number of components (e.g., classes and aspects) that are intended to implement a crosscutting concern and the number of other components that access the concern. A higher CDC indicates higher degree of concern scattering. DOS is measured based on the variance of the lines of code (LOC) spent for a crosscutting concern. The following equation gives the DOS metric for a concern c. DOS ranges between 0 and 1. A higher DOS indicates a higher degree of concern scattering.

$$
\begin{aligned}
DOS(c) &= 1 - \frac{|T| \sum_t (Concentration(c,t) - 1/|T|)^2}{|T| - 1} \\
where & \\
T &= \{t : t \in all\ components\} \\
Concentration(c,t) &= \frac{\text{LOC spent in component } t \text{ to implement a concern } c}{\text{Total LOC in all components that implement a concern c } c}
\end{aligned}
$$

In this experiment, non-functional properties are considered as crosscutting concerns to measure CDC and DOS. DOS requires the LOC that implements a crosscutting concern because it was originally designed to evaluate textual aspect oriented languages. This experiment counts each feature of FM-SNFPs and each tagged-value of UP-SNFPs as one LOC. T in the above equation (a set of components) contains an aspect, a feature configuration and classes stereotyped with ≪`service`≫ and ≪`connector`≫.

The voting process model in Figure 3 is used to measure CDC and DOC. Listing 7 defines an aspect that weaves four different feature configurations shown in Figure 12. `Default` specifies one-way and in-order message transmission. These non-functional properties are woven into all model elements throughput the voting process. `LowLevelSecurity` specifies message signature and logging. They are woven into the model elements that involve in a discussion that the `Moderator` moderates. `HighLevelSecurity` specifies extra security properties in addition to `LowLevelSecurity`. They are woven into the model elements that involves in message flows initiated by `Voting Member`s. `ReliableMessaging` specifies retransmission, integrity, delivery assurance and queuing properties for message transmissions. They are woven into the model elements that involves in communication between `Vote` and `Vote Announcement`.

Table 5 shows CDC of a BPMN model with a BALLAD aspect and a UML model generated by Ark.bpmn. When a BALLAD aspect is used in a business process, CDC is always 2; an aspect and its corresponding feature configuration. In a UML model generated by Ark.bpmn, CDC counts the UML classes that specify non-functional properties and the classes stereotyped with ≪`service`≫ and ≪`connector`≫. As Table 5 illustrates, BALLAD reduces CDC two to eight times by encapsulating non-functional properties into a single aspect.

Table 6 shows DOS of a BPMN model with a BALLAD aspect and a UML model generated by Ark.bpmn. When BALLAD is used for a business process, all non-functional properties are specified in a single aspect. No other model elements specify non-functional properties. As a result, DOS remains low. In a UML model generated by Ark.bpmn, DOS is consistently higher with all

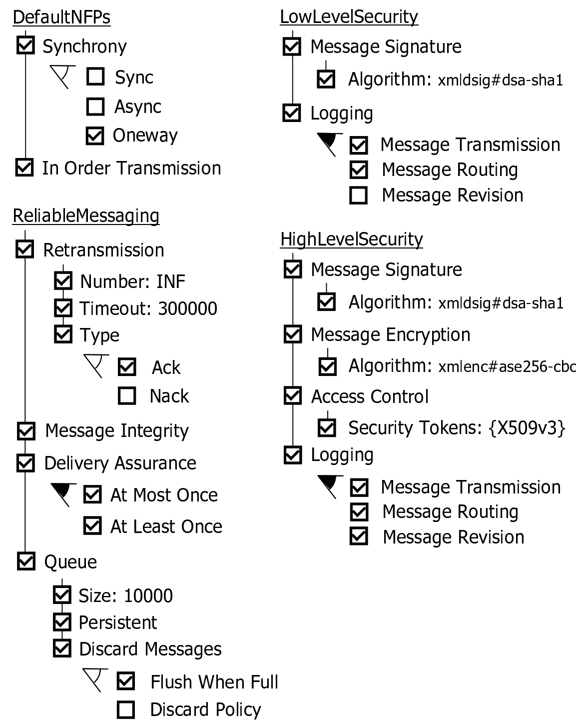

Figure 12. Feature Configurations for a Voting Process Model

Table 5: CDC Measurement

Feature Configuration	BPMN with BALLAD	UML with UP-SNFPs
Default	2	16 (4 services and 12 connectors)
LowLevelSecurity	2	14 (3 services and 11 connectors)
HighLevelSecurity	2	4 (2 services and 2 connectors)
ReliableMessaging	2	7 (3 services and 4 connectors)

of four feature configurations because non-functional properties scatter over UML classes. Eaddy et al. (2007) claim that DOC is 0.5 when crosscutting concerns are well modularized. DOS is close enough to 0.5 when using BALLAD; BALLAD better modularizes non-functional properties than UML models with UP-SNFPs.

As described in the Introduction section, higher degree of concern scattering increases the complexity of application development and maintenance. Figures 5 and 6 demonstrate that BALLAD can significantly reduce the burdens/costs to specify, implement and maintain non-functional properties in SOA.

Performance Measurement

Table 7 shows the overhead to execute each processing step in Ark.bpmn when it transforms the BPMN model in Figure 3 to a UML model by weaving the aspect in Listing 7. In this transformation process, Ark.bpmn (1) parses a given BALLAD aspect, (2) loads a BPMN model, (3) finds paths in a BPMN model according to the aspect, (4) transforms the BPMN model into a UML

Listing 7 An Aspect used for an Empirical Evaluation

```
aspect NFPsForVoting{
 // pointcuts
 pointcut wholeProcess: within("Secretary::Start.*", "Secretary::Finish");
 pointcut discussion: within("Moderator::Discussion.*", "Moderator::Finish");
 pointcut fromMembers: source("Voting Members") && depth(1);
 pointcut voting: within("Vote", "Vote Announcement.*");
 // references to advices
 wholeProcess: Default;
 discussion: LowLevelSecurity;
 fromMembers: HighLevelSecurity:
 voting: ReliableMessaging;
}
```

Table 6: DOS Measurement

Feature Configuration	**BPMN with BALLAD**	**UML with UP-SNFPs**
Default	0.52	0.98
LowLevelSecurity	0.53	0.95
HighLevelSecurity	0.49	0.68
ReliableMessaging	0.43	0.85

model, and (5) configures non-functional properties in the generated UML model with FM-SNFPs. As Table 7 shows, the execution overhead of Ark.bpmn is small enough and acceptable even when it processes a fairly complex BPMN model.

Table 7: Execution Overhead of Each Processing Step in Ark.bpmn

Processing Step	**Time (seconds)**	**Percentage (%)**
(1) Parse an aspect	3.19	30.3
(2) Load a BPMN model	1.41	13.4
(3) Find paths in a BPMN model according to an aspect	0.02	0.2
(4) Transform a BPMN model into a UML model	4.44	42.3
(5) Configure a UML model based on FM-SNFPs	1.45	13.8
Total	10.50	100.0

In order to evaluate the scalability of Ark.bpmn, Figure 13 shows how its execution overhead changes as the size of an input BPMN model grows. Figure 14 illustrates an input BPMN model. As shown, the model size grows by repeatedly adding participants (`Server 01` to `Server N`). This experiment uses the pointcut `within(".*", ".*")`, which takes the longest time among all types of pointcuts, and weaves the `Default` feature configuration in Figure 12 into a input BPMN model. As Figure 13 shows, the total overhead increases at least linearly as the size of an input BPMN model grows. The increase in model size impacts the overhead of Step 3 and 4; however, the two steps remain reasonably lightweight. The increase has few or no impacts on the overhead of the other three steps. Figure 13 demonstrates that Ark.bpmn scales well against large-scale BPMN models.

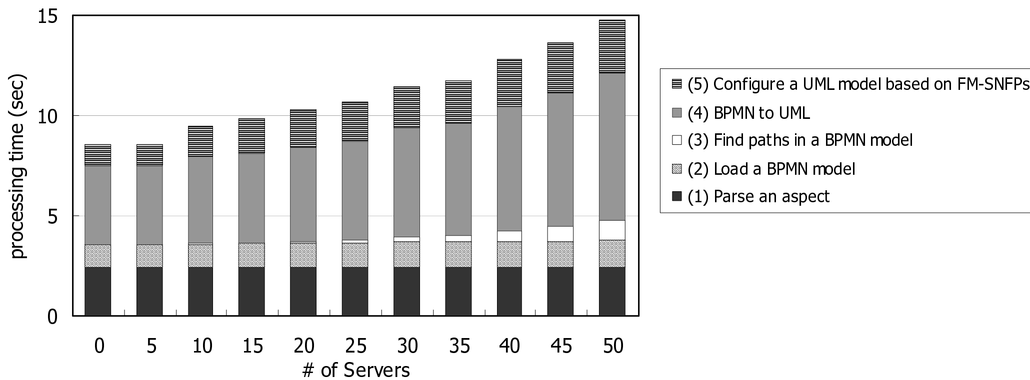

Figure 13. Execution Overhead of Ark.bpmn with BPMN models of Different Sizes

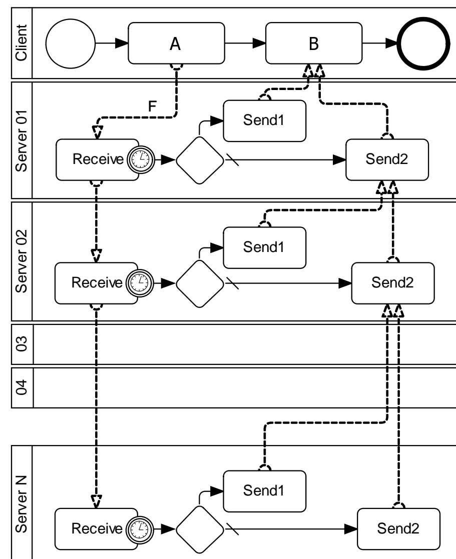

Figure 14. An Example BPMN Model for a Scalability Evaluation

In addition to the size of an input BPMN model, different types of pointcuts can impact the overhead of Step 3 differently. Figure 15 shows how the overhead of Step 3 changes with 8 different pointcuts. `within(".*", ".*")` and `target("X")` are the two slowest pointcuts; however, they are still lightweight enough. It takes only 1.2 and 0.4 seconds to execute the two pointcuts. The other six pointcuts are very fast, and their overhead can be even negligible. Figure 15 demonstrates that BALLAD is designed and implemented lightweight.

Related Work

This paper focuses on a set of extensions to the authors' previous work (?, ?, ?). The previous work studied non-functional properties in UML models on a per-service basis. In contrast, this paper considers non-functional properties in business processes and proposes an aspect oriented language for a new per-process strategy to separate functional and non-functional properties in service oriented applications. Moreover, this paper considers the transformation from implementation

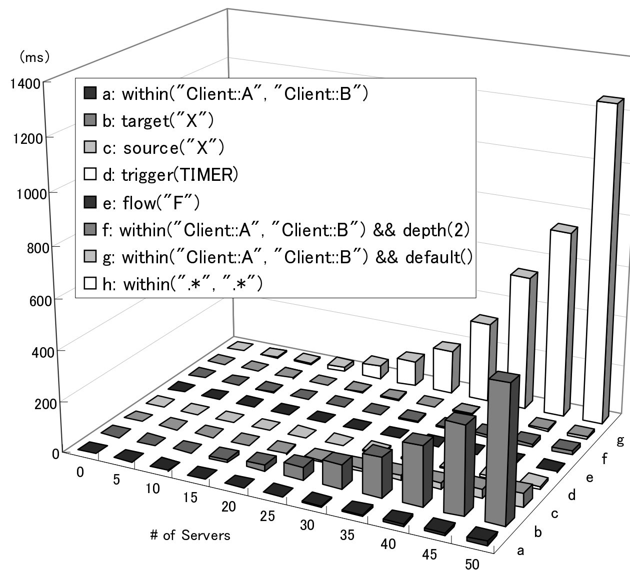

Figure 15. Execution Overhead of Finding Paths

independent to implementation specific UML models, which were beyond of the scope of the previous work.

AspectViewpoint is an aspect oriented language to define aspects for BPMN models (Correal & Casallas, 2007). It uses aspects to define business processes, and extends an existing business process by weaving the new ones to it. For example, when a purchasing business process is defined as a BPMN model, a new business process (e.g., order cancellation process) can be defined as an aspect and woven to the purchasing process for considering order cancellation in purchasing. AOPML is an aspect oriented language to extend business processes by weaving new tasks into it (Julio Leite, Batista, & Silva, 2009). Courbis and Finkelstein (2005) propose an aspect oriented language to specify aspects for business processes defined in Business Process Execution Language (BPEL) (*Web Services Business Process Execution Language*, 2003). 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. For example, an aspect may be defined to insert a `<switch>` block, which performs a branching operation in BPEL, right before `<invoke>` blocks, which are used to invoke services. This aspect modifies the semantics of service invocation by considering a certain condition(s). The above three languages are similar to BALLAD in that they study aspects for business processes. However, they focus on functional properties of business processes. Unlike them, BALLAD modularizes non-functional properties of business processes as aspects.

AO4BPEL is an aspect oriented language to extend BPEL business processes with non-functional properties such as reliable messaging, message encryption and transactions (Charfi, Schmeling, Heizenreder, & Mezini, 2006). Aspects can specify non-functional properties that are woven to services and their activities/tasks; however, a variety of pointcuts is limited in AO4BPEL. BALLAD offers higher expressiveness in defining aspects; it considers the pointcuts in control/message flows as well as tasks and pools. Also, it supports much more non-functional properties than AO4BPEL does.

Zou, Xiao, and Chan (2007) propose an aspect oriented language to weave non-functional

properties to BPEL business processes. However, it does not provide specific non-functional properties and does not perform code generation. In contrast, BALLAD provides a specific set of non-functional properties in an unambiguous manner. BALLAD aspects are directly used for code generation.

Aburub, Odeh, and Beeson (2007) propose 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 which business processes. However, Aburub et al. (2007) does not provide a language to explicitly declare non-functional requirements in business processes. As a result, the synthesis of functional and non-functional properties is manually performed. Code generation is not supported either. In contrast, BALLAD is intended to specify non-functional properties, which are adjustable parameters used for satisfying given non-functional requirements. BALLAD can explicitly express how to weave non-functional properties to business processes. Ark implements code generation for BALLAD aspects.

Xu, Ziv, Richardson, and Liu (2005) propose 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 consider code generation. In contrast, BALLAD defines a set of non-functional properties as an aspect and weave it to business processes. It also provides a specific set of non-functional properties in an unambiguous manner and supports code generation for them.

Several modeling languages (e.g., UML profiles and domain-specific languages) have been proposed to specify non-functional properties in SOA, such as security, data integration, service discovery and service orchestration (?, ?; Amir & Zeid, 2004; Ortiz & Hernändez, 2006; L. Wang & Lee, 2005; Nakamura et al., 2005). However, they do not focus on modeling a series of constraints among those non-functional properties. FM-SNFPs provides a method to explicitly model non-functional constraints in SOA and automatically enforce the constraints in their applications.

The notion of feature modeling has been used to, for example, configure the non-functional policies in embedded operating systems (e.g., concurrency and interruption policies) (Lohmann, Scheler, Preikschat, & Spinczyk, 2006), configure the functionalities of Eclipse plugins (e.g., multi-windows) (Antkiewicz & Czarnecki, 2006), configure underlying platform technologies (e.g., databases) used in applications (White et al., 2007) and select services in PBX systems (e.g., call request and call forwarding services) (Kang, Kim, Lee, & Lee, 1999). FM-SNFPs leverages feature modeling for managing non-functional constraints in SOA.

Czarnecki and Antkiewicz (2005) propose feature-based model templates designed to transform feature configurations to UML models. Each template is a UML class or activity diagram that defines a *presence condition* for each model element (e.g., class, association and action). A presence condition specifies when a corresponding model element appears in a UML diagram. For example, a class may have its presence condition `Async & Ack` in a class diagram template. The template generates the class in an output class diagram if the template accepts an input feature configuration that selects both `Async` and `Ack` features. This way, non-functional properties scatter both in feature configurations and model templates. This makes it complicated to maintain non-functional properties; a change (e.g., addition, removal or customization) in non-functional properties always require changing both feature models and model templates. In contrast, FM-SNFPs modularizes non-functional properties in feature configurations;

non-functional properties never appear in input BPMN models.

Kaul, Kogekar, Gokhale, Gray, and Gokhale (2007) propose a domain specific language, called POSAML (Pattern Oriented Software Architecture Modeling Language), to visually configure non-functional properties in realtime CORBA middleware (e.g., concurrency and message queuing). In POSAML, non-functional constraints are specified in a textual form with Object Constraint Language (OCL) (Warmer & Kleppe, 2003). Although OCL provides higher expressiveness as a constraint specification language than feature models, FM-SNFPs employs feature models by trading visual intuitiveness for expressiveness.

Several methods exist to explicitly specify, examine and enforce non-functional requirements. Robinson and Purao (2009) propose an extension to OCL for defining functional and non-functional requirements in the interactions among services. For example, the language can specify, as a functional requirement, that a buyer receives a receipt within five minutes after a purchase. A monitoring tool examines whether given requirements are satisfied at runtime and notifies when a violation occurs. Chung, Nixon, Yu, and Mylopoulos (1999) propose a feature modeling method to define non-functional requirements and analyze the relationships among them. For example, the method can specify security and performance requirements and define their sub-requirements (e.g., integrity and confidentiality for the security requirement). It aids identifying and analyzing a potential conflict relationship between the integrity sub-requirement and performance requirement. The goals of these work and BALLAD are different. BALLAD focuses on generic non-functional properties and maps them to low-level implementation technologies such as transport protocols and remoting middleware, while Robinson and Purao (2009) and Chung et al. (1999) focus on application-specific non-functional requirements and does not consider their realization with implementation technologies. Moreover, they do not consider to modularize scattering non-functional requirements with the notion of aspects.

Conclusion

This paper investigates an end-to-end MDD framework that manages non-functional properties in SOA from high-level business processes to low-level configurations of implementation technologies such as transport protocols and remoting middleware. This framework proposes (1) an aspect oriented language, called BALLAD, for a new per-process strategy to specify non-functional properties in business processes, and (2) a graphical modeling method, called FM-SNFPs, to explicitly specify non-functional constraints and consistently validate and enforce them in applications. Empirical evaluation results show that BALLAD significantly reduces the costs to implement and maintain non-functional properties in SOA. BALLAD and FM-SNFPs are designed and implemented efficient and scalable.

Acknowledgement

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

References

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

Allcock, W., Bresnahan, J., Kettimuthu, R., M. Link, C. D., Raicu, I., & Foster, I. (2005, November). The globus striped gridftp framework and server. In *Super computing*.

Amir, R., & Zeid, A. (2004, October). A uml profile for service oriented architectures. In *Acm sigplan conference on object-oriented programming, systems, languages, and applications, poster session.*

Antkiewicz, M., & Czarnecki, K. (2004, October). Featureplugin: Feature modeling plug-in for eclipse. In *Acm sigplan conference on object-oriented programming, systems, languages, and applications, workshop on eclipse technology exchange.*

Antkiewicz, M., & Czarnecki, K. (2006, October). Framework-specific modeling languages with round-trip engineering. In *Acm/ieee international conference on model driven engineering languages and systems.*

Baligand, F., & Monfort, V. (2004, December). A concrete solution for web services adaptability using policies and aspects. In *Acm sigsoft/acm sigweb international conference on service oriented computing.*

Bichler, M., & Lin, K. (2006, June). 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.

Business process modeling notation (bpmn) 1.0. (2004). Business Process Modeling Initiative.

Charfi, A., Schmeling, B., Heizenreder, A., & Mezini, M. (2006, December). Reliable, secure, and transacted web service compositions with ao4bpel. In *Ieee european conference on web services.*

Chitchyan, R., Rashid, A., Sawyer, P., Garcia, A., Alarcon, M. P., Bakker, J., et al. (2005, May). *Survey of aspect-oriented analysis and design approaches* (Tech. Rep.). AOSE-Europe.

Chung, L., Nixon, B., Yu, E., & Mylopoulos, J. (1999). *Non-functional requirements in software engineering*. Kluwer Academic Publishers.

Correal, D., & Casallas, R. (2007, October). Using domain specific languages for software process modeling. In *Acm sigplan conference on object-oriented programming, systems, languages, and applications, workshop on domain-specific modeling.*

Courbis, C., & Finkelstein, A. (2005, October). Weaving aspects into web service orchestrations. In *Ieee international conference on web services.*

Czarnecki, K., & Antkiewicz, M. (2005, September). Mapping features to models: A template approach based on superimposed variants. In *International conference on generative programming and component engineering.*

Czarnecki, K., & Eisenecker, U. (2000). *Generative programming: Methods, tools and applications*. Addison-Wesley.

Czarnecki, K., Helsen, S., & Eisenecker, U. (2005). Formalizing cardinality-based feature models and their specialization. *Software Process Improvement and Practice*, *10*(1).

Eaddy, M., Aho, A., & Murphy, G. C. (2007, May). Identifying, assigning, and quantifying crosscutting concerns. In *Aosd-europe international workshop on assessment of contemporary modularization techniques.*

Elrad, T., Aldawud, O., & Bader, A. (2002, October). Aspect-oriented modeling - bridging the gap between design and implementation. In *Acm international conference on generative programming and component engineering.*

Erickson, J., & Siau, K. (2008). Web services, service-oriented computing, and service-oriented architecture: Separating hype from reality. *Journal of Database Management*, *19*(3), 42–54.

Jřjens, J. (2002, October). Umlsec: Extending uml for secure systems development. In *Acm/ieee international conference on unified modeling language.*

Julio Leite, C. C. an, Batista, T., & Silva, L. (2009, March). An aspect-oriented approach to business process modeling. In *Acm international conference on aspect-oriented software development, workshop on early aspects.*

Kang, K., Kim, S., Lee, J., & Lee, K. (1999, December). Feature-oriented engineering of pbx software. In *Asia-pacific software engineering conference.*

Kaul, D., Kogekar, A., Gokhale, A., Gray, J., & Gokhale, S. (2007, January). Posaml: A visual modeling framework for middleware provisioning. In *Hawaiian international conference on system sciences.*

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. M., et al. (1997, June). Aspect-oriented programming. In *European conference on object-oriented programming.*

Lodderstedt, T., Basin, D., & Doser, J. (2002, October). Secureuml: A uml-based modeling language for model-driven security. In *Acm/ieee international conference on unified modeling language.*

Lohmann, D., Scheler, F., Preikschat, W. S., & Spinczyk, O. (2006, July). Pure embedded operating systems - ciao. In *Ieee international workshop on operating system platforms for embedded real-time applications.*

Nakamura, Y., Tatsubori, M., Imamura, T., & Ono, K. (2005, July). Model-driven security based on a web services security architecture. In *Ieee international conference on services computing.*

Ortiz, G., & Hernändez, J. (2006, September). Toward uml profiles for web services and their extra-functional properties. In *Ieee international conference on web services.*

Papazoglou, M., & Heuvel, W. (2007, July). Service oriented architectures: Approaches, technologies and research issues. *The International Journal on Very Large Data Bases*, *16*(3), 389–415.

Robinson, W. N., & Purao, S. (2009). Specifying and monitoring interactions and commitments in open business processes. *IEEE Software*, *26*(2), 72–79.

Sant'Anna, C. N., Garcia, A. F., Chavez, C., Lucena, C. J. P., & Staa, A. (2003, October). On the reuse and maintenance of aspect-oriented software: An assessment framework. In *Brazilian symposium on software engineering.*

Soler, E., Villarroel, R., Trujillo, J., Medina, E. F., & Piattini, M. (2006, April). Representing security and audit rules for data warehouses at the logical level by using the common warehouse metamodel. In *International conference on availability, reliability and security.*

Vokäc, M. (2005, October). Using a domain-specific language and custom tools to model a multi-tier service-oriented application–experiences and challenges. In *Acm/ieee international conference on model driven engineering languages and systems.*

Wang, G., Chen, A., Wang, C., Fung, C., & Uczekaj, S. (2004). Integrated quality of service (qos) management in service-oriented enterprise architectures. In *Ieee enterprise distributed object computing conference.*

Wang, L., & Lee, L. (2005). Uml-based modeling of web services security. In *Ieee european conference on web services poster session.*

Warmer, J., & Kleppe, A. (2003). *The object constraint language: Getting your models ready for mda* (Second ed.). Addison-Wesley.

Web services business process execution language. (2003). OASIS.

White, J., Schmidt, D., Czarnecki, K., Wienands, C., Lenz, G., Wuchner, E., et al. (2007, October). Automated model-based configuration of enterprise java applications. In *Ieee international conference on enterprise distributed object computing conference.*

Xu, L., Ziv, H., Richardson, D., & Liu, Z. (2005, March). Towards modeling non-functional requirements in software architecture. In *Acm international conference on aspect-oriented software development early aspects workshop.*

Zou, Y., Xiao, H., & Chan, B. (2007, September). Weaving business requirements into model transformations. In *Acm/ieee international conference on model driven engineering languages and systems, workshop on aspect-oriented modeling.*

Footnotes

[1]Business Process Aware Language for earLy Aspect Design

[2]http://www.eclipse.org/emf/

[3]http://gp.uwaterloo.ca/fmp/

[4]http://www.soyatec.com/ebpmn/

[5]This model is made based on an example model included in the BPMN specification (*Business Process Modeling Notation (BPMN) 1.0*, 2004)

[6]An extension to FTP for transmitting files of large size (Allcock et al., 2005).

[7]http://www.openarchitectureware.com/

[8]http://mule.codehaus.org/
[9]http://servicemix.apache.org/