Using Aspect Oriented Techniques to Support Separation of Concerns in Model Driven Development

Arnor Solberg
SINTEF
PB 124 Blindern, 0314 Oslo, Norway
arnor.solberg@sintef.no

Devon Simmonds, Raghu Reddy, Sudipto Ghosh, Robert France
Department of Computer Science
Colorado State University Fort Collins, CO – 80523
{simmonds,raghu,ghosh,france}@cs.colostate.edu

Abstract

Model driven development (MDD) tackles software complexity through the use of models. However, managing relationships and specifying transformations between models at various levels of abstraction are complex tasks. System models tangled with concerns such as security and middleware make it difficult to develop complex systems and specify model transformations. This paper presents an MDD framework that uses aspect oriented techniques to facilitate separation of concerns. We argue that using the framework will simplify both the model development task and the task of specifying transformations. The conceptual model of the framework is presented and illustrated using distributed transactions at the PIM and PSM levels.

Keywords: *model driven development, aspect-oriented development, middleware, separation of concerns.*

1. Introduction

Model driven development (MDD) shifts software development from a code-centric activity to a model-centric activity. Accomplishing this shift entails support for modeling concepts at different levels of abstraction and transforming abstract models to more concrete descriptions of software. In MDD, mechanisms for both vertical and horizontal separation of concerns should be provided for reducing complexity. Vertical separation of concerns reduces complexity through abstraction and horizontal separation of concern reduces complexity by describing the system using manageable system views.

The model driven architecture (MDA) [1] initiative proposes a system to be modeled at three different levels of abstraction (vertical separation): computation independent model (CIM), platform independent model (PIM) and platform specific model (PSM). Separation of PIM and PSM occurs when one defines a middleware technology independent model and a middleware technology dependent model. In this paper we focus on transforming models from PIM to PSM.

A Mechanism for separation of concerns at the same abstraction level (*horizontal separation*) is to model the system using views. A system view describes a certain facet of the system (e.g., structure, behavior or distribution). Generic frameworks such as the ISO RM-ODP framework [2] may be used. Views may be specified using different diagram types provided by a modeling language. A diagram type provides a set of constructs for specifying a specific view of the system (e.g., UML [3] provides activity, class and state diagrams). Modeling languages such as UML and frameworks such as RM-ODP provide a fixed set of views. Diagram types only provide separation of structure and behavior. Neither generic view frameworks nor diagram types inherently provide separation of concerns like security and transactions that are tangled and scattered with the business functionality. Such scattered and tangled concerns are referred as crosscutting concerns. To manage crosscutting concerns, they need to be isolated from the other views. Aspect Oriented Software Development (AOSD) [5][6][7] provides the mechanisms for encapsulating crosscutting concerns using *aspects*. In our Aspect Oriented Modeling approach [13][14] crosscutting concerns are explicitly modeled as aspects and woven with the primary model that specifies the business logic of the application.

To derive the system at the PSM level, specification of the mappings from PIM to PSM are needed. However, specifying mappings of a system specification from the PIM level to a system specification at the PSM level may be a complex task when the PIM is a composite model of all software concerns. The baseline for the framework is to model each crosscutting feature separately at the PIM level using aspect oriented techniques. For PIM to PSM model transformation specifications separate mappings for the primary model and each of the aspects are defined.

2. Background

2.1 Model Transformation

One major goal of the MDD approach is to specify the software system at a platform independent level, and derive the system at the platform specific level as indicated in Figure 2. Model transformation is performed based on the mapping specification from PIM to PSM.

Most model transformation approaches are based on specifying mappings from source meta-model concepts to the target meta-model concepts, as well as deriving target patterns based on source pattern recognition [18][19]. However, simple meta-model mappings may not deliver the desired results. For instance, it may not be desirable to map all instances of a specific meta-model element at the PIM level the same way. Depending on the context at the PSM level, it may be necessary to let instances of the same metamodel element appear differently at the PSM level. Moreover, to derive a PSM, facilites provided by the platform as well as recommended patterns and practices need to be utilized. For instance, most middleware platforms provide specific services for handling security, persistence, and transactions. These services typically require specific protocols to be followed. The paper will illustrate that these cases need to be treated separately to obtain the desired result.

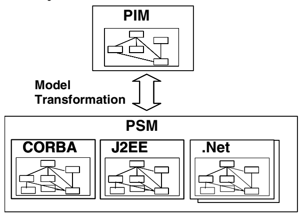


Figure 1. MDD vision

2.2 Aspect Oriented Modeling (AOM)

There is ongoing research that investigates how to apply AOSD techniques at the model level [8][13][14][16]. Our Aspect Oriented Modeling (AOM) approach [13][14] focuses effort on separation of concerns at the model level. The AOM approach specifies: 1) A *primary model* that addresses the business logic of the application, 2) A set of *generic aspect models*, where each model is a generic description of a crosscutting concern. The model elements are expressed as roles, 3) A set of bindings that determines where in the primary model the aspect models are to be composed, 4) A set of composition directives that influence how aspect models are composed with the primary model.

Before an aspect model can be composed with a primary model in an application domain, the aspect model must be instantiated in the context of the application domain. An instantiation is obtained by binding elements in the aspect model to elements in the application domain. The result is called a context-specific aspect model. Context-specific aspect models and the primary model are composed to obtain an integrated design view.

3. Conceptual Framework

The proposed framework helps focus effort on developing solutions for individual crosscutting concerns. The conceptual framework is shown in figure 2.

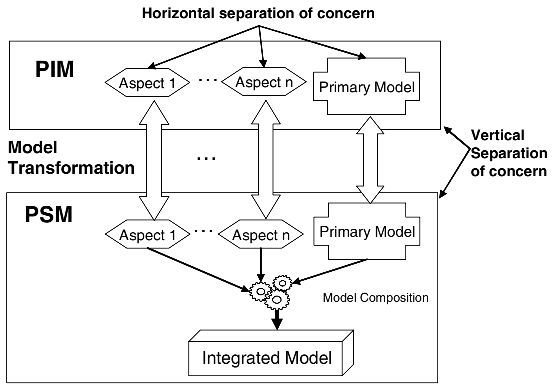


Figure 2. Conceptual model

At the PIM level, a primary model and aspect models are developed. The primary model encapsulates the application structure and business functionality, while the aspect models encapsulate the aspects. The transformation between the PIM and the PSM is defined by separate mappings for each aspect and the primary model. The aspect and mappings are obtained based on the type of middleware and the protocols defined by the middleware. The proposed framework simplifies software modeling and the specification of transformations from PIM to PSM for the following reasons: 1) The framework allows developers to conceptualize, describe, and communicate crosscutting concerns as conceptual units at various levels of abstraction, 2) The horizontal separation of concerns as aspect models and a primary model facilitate separate specification of mappings, 3) The specification of the transformation of an aspect or the primary model from PIM to PSM is less complex than the specification of the transformation of an integrated PIM model to PSM, since the latter transformation is likely to have more relationships and dependencies., 4) Changes to a crosscutting concern can be made in one place, and effected by composing the changed aspect model with a primary model.

The platform specific models are obtained by applying the PIM to PSM transformations At the PSM level an integrated model view of the system is obtained by composing the primary model and the context-specific aspect models. Model composition is performed according to defined composition directives (see [13]). Model composition may also be performed at the PIM level since the composed model can be used for conformance checking of the composed PSM model.

4. Illustration of the Framework

In this section we illustrate the framework with a distributed banking application that offers electronic money transfer using distributed transaction services. An example of the PIM to PSM aspect transformation is also presented here.

4.1 A Banking Scenario

The bank consists of a set of accounts and customers. The business functionality includes operations to open and close accounts and to add and delete customers. Withdrawal and deposit of specific amounts of money are provided. The transfer of money requires transaction control which is modeled as an aspect. The money transfer scenario is shown as an activity diagram in figure 3.

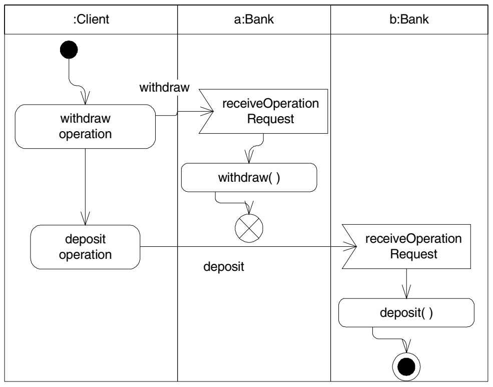


Figure 3: Money transfer scenario primary model

5. Generic Transaction Aspect

A transaction is an indivisible collection of operations between servers and clients that remain atomic even when some clients and servers fail. An atomic operation is an operation that is free of interference from concurrent operations. Transactions are required to manifest the `ACID' properties [17]. The intent of these properties is to assure the trustworthiness of transactions. Different middleware may provide different transaction protocols and each protocol may be implemented in a different manner.

The UML 2.0 activity diagram of Figure 4 shows a platform independent generic transaction aspect for the two-phase commit distributed transaction protocol. The aspect has three main roles: 1) A **Transaction Client** initiates the transaction and performs a collection of operations for the specific transaction, 2) A **Participant** provides some service required by the client or another participant, 3) A **Transaction Manager** responsible for coordinating and managing transactions .

The Transaction Client initiates the transaction by sending the initiateTransaction When the Transaction Manager receives initiateTransaction, it opens a transaction and returns a transaction id (Tid). This Tid is sent as a parameter for all subsequent operations. The Transaction Client then performs the collection of operations of the transaction. When a Participant receives an operation request it checks whether it is already a member of the particular transaction. If not it joins the transaction, before it performs the requested operation.

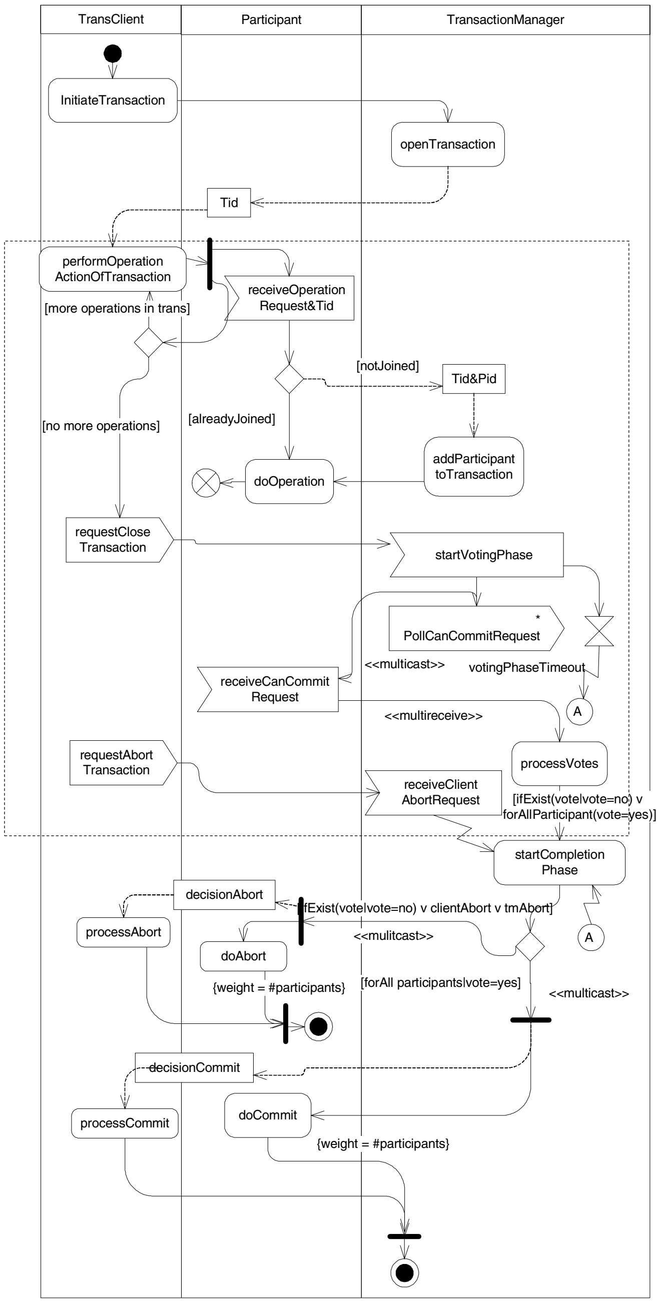


Figure 4: Aspect Transaction, PIM level two phase commit protocol

Two-Phase Commit Protocol: When the transaction client requests to close the transaction, the Transaction Manager starts the two phase commit protocol. In the first phase (*voting phase*), the transaction manager polls the participants to determine if they are ready to commit. In the second phase (*completion phase*), the Transaction Manager decides to abort or commit the specific transaction. The decision is multicast to all participants. The interruption region shown in the dotted square includes two possible interruptions (interruptions are indicated with a lightening arrow symbol). Either the transaction clients request to abort the transaction or the transaction manager exceeds the timeout limit for the voting phase. Both interruptions cause any flow in the interruption region to stop, and the completion phase is initiated. As the result, the decision will then be to abort the transaction.

A participant may decide to abort at any time during the transaction. The two phase commit will then eventually decide to abort and all participants will be informed and the participants will roll back the transactions. According to the general two phase commit protocol a participant may request the status of the transaction at any time and decide whether it wants to continue with the transaction (this is not shown in the figure).

5.3 Distributed Transactions in CORBA

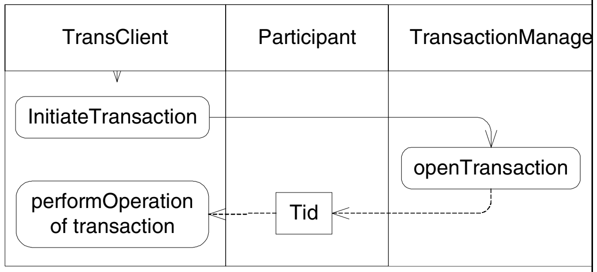
In CORBA, distributed transactions are facilitated through the object transaction service (OTS) model. The OTS has three types of application objects: transactional clients, transactional servers and recoverable servers. A transactional client is a client application that creates a transaction and invokes operations on transactional objects provided by the server. The server object and all resource objects associated with the server that are affected by the transaction are called transactional objects. A transactional server contains one of more transactional objects. A recoverable object is a transactional object that directly manages persistent data. Recoverable objects participate in the transaction completion protocol by creating and registering resource objects with the transaction service. A recoverable server contains one or more recoverable objects. The transaction service manages the two phase commit protocol for all registered resources.

PIM to PSM Mapping: Table 1 provides an example of one of the PIM to PSM mappings. Table 1(a) shows a PIM fragment that initiates the transaction. It consists of three activities and a data flow. Table 1(b) shows a PSM fragment realized by making the following mappings: 1) The *initiateTransaction* activity in the PIM is mapped to the *initiateTransaction* activity in the PSM, 2) The *openTransaction* to *performOperation of transaction* data flow in the PIM is mapped to the *createNewTransaction* to *performOperation of transaction* data flow in the PSM, 3) The *transactionManager* in the PIM is mapped to *CurrentHelper* and *Current* in the PSM

1(c) gives a second mapping for the same PIM fragment. This mapping accommodates use of the CORBA *TransactionFactory* interface. This mapping produces a different sequence of activities and data flow than that in Table 1(b).

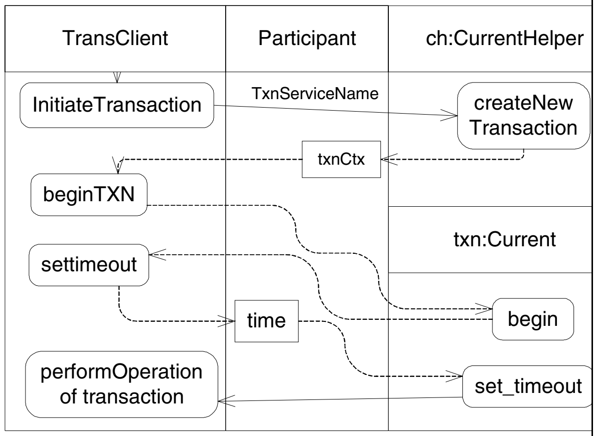
Table 1: Composite PIM to PSM Mappings

PIM ITEM

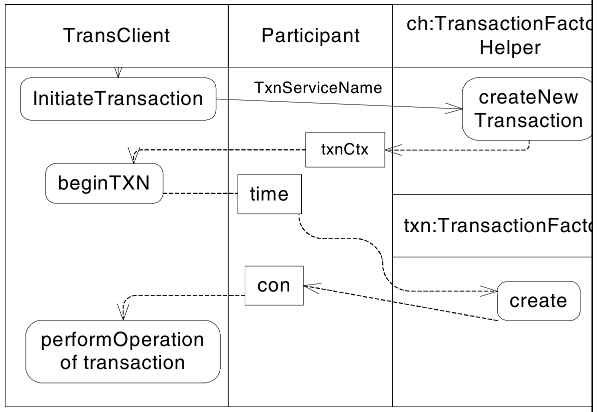


(a) Initiate transaction data flow

PSM ITEMS



(b) Initiate transaction data flow using the CORBA *Current* interface



(c) Initiate transaction data flow using the CORBA *TransactionFfactory* interface

The table illustrates why a meta-model to meta-model transformation may not provide the desired results for middleware protocols.

The mapping example shown in table 1 helps illustrate the following points: 1) Entities in the PIM and those in the PSM do not necessarily bear a one-to-one relationship.

One item in the PIM may be mapped to a multiple item in the PSM and vice versa. For example the *TransactionManager* role in Figure 3 is mapped to four roles in Figure 4: *CurrentHelper, Current, Control* and *Coordinator*, 2) The PSM generated for an application may depend on design decisions. For example a decision to use implicit transaction context propagation will result in the mapping in Table 1(b) while explicit transaction context propagation results in Table 1(c), 3) Items in the PIM may be absent in the PSM. For example the *haveComitted* activity in Figure 3 has no equivalent CORBA activity in Figure 4, 4) The PSM may also introduce new items which are absent from PIM.

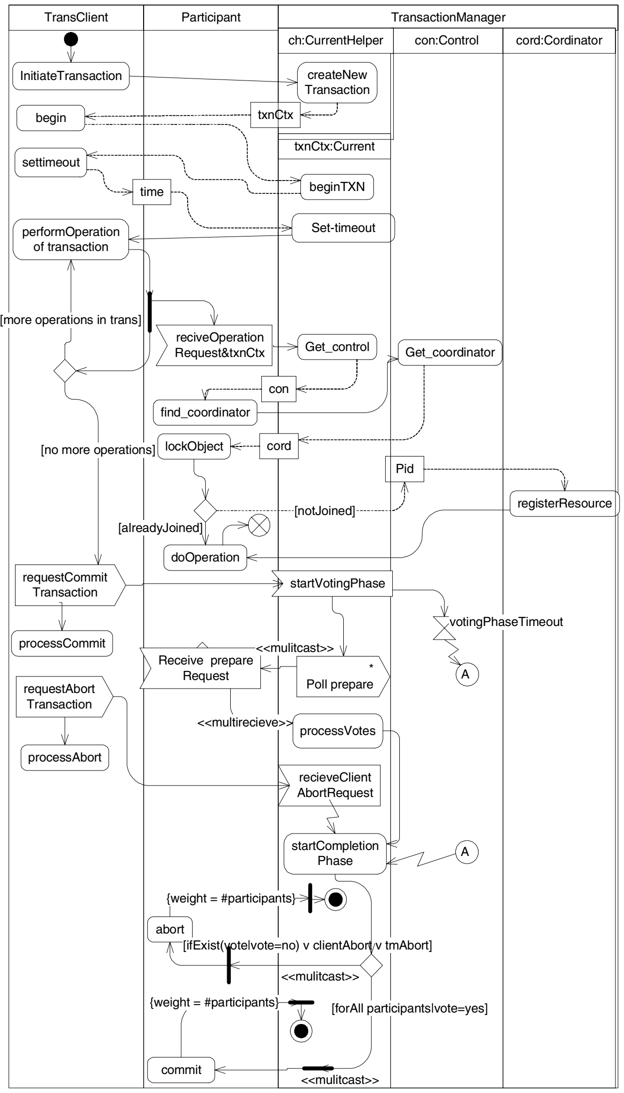


Figure 5. CORBA Transaction Aspect

Figure 5 shows a CORBA PSM transaction aspect. The PSM aspect was realized by mapping items in the PIM to items in the PSM.

5.4 Composition

The CORBA, distributed transaction aspect presented in Figure 5 can be composed with a platform specific activity diagram that captures a primary model money transfer scenario. For the composition to occur, the aspect model has to be instantiated in the context of the primary model. Because aspects are generic, instantiations provide information about the model elements in the generic platform specific aspect model that need to be bound to model elements in the platform specific primary model. For example, some of the instantiations are: 1) *TransClient* in Figure 5 is bound to Client in a platform specific money transfer scenario, 2) *Participant* is mapped twice, once for *a:Bank* and once for *b:Bank*, 3) *PerformOperation* of Transaction is mapped to the *withdrawOperation* and *depositOperation*.

Similarly other bindings are specified. Once the platform specific aspect models are instantiated in the context of the application, they can be composed to produce an integrated view of the system. The composition mechanism has been published in our previous papers [13][15].

6. Related work

Jacobson [10][11] describes the development of design aspects based on use cases, which are then composed to create different views of the system. The work maps directly to program level aspects, using the composition techniques originally developed for AspectJ [4]. The work does not explicitly give details about transformation of models, rules of composition, structural relations, etc.

Reina et al. [9] propose the use of meta-models and UML profiles for separation of concerns at the PIM and PSM levels. The problem with this approach lies in using different meta-model for every new concern. In the aspect-oriented modeling approach proposed by Clarke et al. [16], a design called a subject is created for each system requirement. A Comprehensive design is a composition of subjects. Subjects are expressed as UML model views, and composition merges the views provided by the subjects. The approach does not consider any middleware aspects.

Kulkarni et al. [12] present a model driven development approach for separation of concerns. They use an abstract template to separate system concerns at the model and code levels. This is similar to our AOM approach. The AOM approach uses parameterized UML to specify aspects. In addition, AOM uses parameterized OCL to perform verifiable composition [15].

7. Conclusion and Further Work

Modern systems are complex. Separation of concerns is recognized as a key principle to cope with complexity in software development. In this paper, we have reasoned that both vertical and horizontal separation of concerns should be provided, for managing complexity in a model driven development.

Aspect-oriented technologies can be used to support horizontal separation of crosscutting concerns from other functionality. The AOM approach emphasizes the separation and modularization of crosscutting concerns in design units (aspects). Also, the AOM approach is model driven and specifies the transformations from a platform independent model to a platform specific model.

Currently we are working on techniques to resolve conflicts, if more than one aspect needs to be composed with the primary model. Verifiable composition techniques that discharge proof obligations during composition are being developed. We also plan to apply different middleware mappings to the same transaction and determine the feasibility of the approach. Also, we plan to create a repository of the most common middleware concerns.

8. References

[1] OMG MDA™ Guide v1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf

[2] ISO/IEC 10746: (1995): Basic reference model for open distributed processing

[3] OMG, Unified Modeling Language (UML™) 1.5 Specification, Object Management Group, Document formal/03-03-01, 2003

[4] Eclipse AspectJ project, http://eclipse.org/aspectj/

[5] Aspect Oriented Software Development. AOSD Webpage. URL http://aosd.net/, 2005.

[6] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingier, and J. Irwin. Aspect Oriented Programming. In Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Springer Verlag LNCS 1241, Finland, June 1997.

[7] I. Ray, R. France, N. Li, and G. Georg, "An Aspect-Based Approach to Modeling Access Control Concerns", Journal of Information and Software Technology, 46(9), pp. 575-587, July 2004.

[8] D. C. Schmidt, A. Gokhale, B. Natarajan, S. Neema, T. Bapty, J. Parsons, A. Nechipurenko, J. Gray and N. Wang. CoSMIC: A MDA tool for Component Middleware-based Distributed Real-time and Embedded Applications. Proceedings of OOPSLA Workshop on Generative Techniques for Model-Driven Architecture, Seattle, WA USA, November 2002.

[9] A. M. Reina, J. Toress, and M. Toro. Towards developing generic solutions with aspects. In proceedings of the Workshop in Aspect Oriented Modeling held in conjunction with UML 2004, October 2004.

[10] I. Jacobson. Case for Aspects - Part I. Software Development Magazine, pages 32-37, October 2003.

[11] I. Jacobson. Case for Aspects - Part II. Software Development Magazine, pages 42-48, November 2003.

[12] Vinay Kulkarni, Sreedhar Reddy. Separation of Concerns in Model-driven Development. IEEE Software 20(5):64-69, 2003.

[13] R. B. France, I. Ray, G. Georg, and S. Ghosh. An aspect-oriented approach to design modeling. IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, 151(4), August, 2004.

[14] G. Georg, R. Reddy, and R. France. Specifying cross-cutting requirements concerns. In Proceedings of the International Conference on the UML, October 2004. Springer, 2004.

[15] E. Song, R. Reddy, R. France, I. Ray, G. Georg, R. Alexander. Verifying Access Control Properties using Aspect Oriented Modeling. Submitted to SACMAT 2005.

[16] S. Clarke, W. Harrison, H. Ossher, and P. Tarr. Separating concerns throughout the development lifecycle. In Proceedings of the 3rd ECOOP Aspect-Oriented Programming Workshop, Lisbon, Portugal, June 1999.

[17] Greg Straw, Geri Georg, Eunjee Song, Sudipto Ghosh, Robert France, and James M. Bieman, "Model Composition Directives", accepted to the 7th UML Conference, Lisbon, Portugal, October 10-15, 2004.

[18] Revised submission for MOF2.0 Query/Views/Transformations RFP (ad/2002-04-10), QVT-Merge Group 1.8, OMG document ad/2004-10-04. www.omg.org

[19] Czarnecki.K.,Helsen S., Classiciation of Model Transformation Approaches, Proceedings Workshop on Generative Techniques in the Context of Model-Driven Architecture,OOPSLA'03