An Aspect Oriented Model Driven Framework

Devon Simmonds
Department of Computer Science
Fort Collins, CO – 80523
{simmonds}@cs.colostate.edu

Arnor Solberg
SINTEF
P.O. Box 124 Blindern
N-0314 Oslo, Norway
arnor.solberg@sintef.no

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

Abstract

In model driven development (MDD), specifying transformations between models at various levels of abstraction can be a complex task. Specifying transformations for pervasive system features that are tangled with other system features is particularly difficult because the elements to be transformed are distributed across a model. This paper presents an aspect oriented model driven framework (AOMDF) that facilitates separation of pervasive features and supports their transformation across different levels of abstraction. The framework is illustrated using an example in which a platform independent model of a banking application is transformed to a platform specific model.

Keywords: *aspect-oriented software development, distributed applications, middleware, model driven development, separation of concerns, UML.*

1 Introduction

Model driven development (MDD) shifts software development from a code-centric activity to a model-centric activity. Accomplishing this shift entails developing support for modeling concepts at different levels of abstraction and transforming abstract models to more concrete descriptions of software. MDD aims to reduce complexity in software development through modularization and abstraction.

An MDD framework should provide mechanisms supporting both vertical and horizontal separation of concerns. The model driven architecture (MDA) [1][2] initiative of the OMG proposes a vertical separation of concerns mechanism consisting of three different levels of abstraction: computation independent model (CIM), platform independent model (PIM) and platform specific model (PSM). For example separation of platform independent and platform specific concerns occurs when a middleware independent model (a PIM) and a corresponding middleware specific model (a PSM) are defined for a particular application.

Horizontal separation of concerns is typically realized by modeling a system using views (e.g., the ISO RM-ODP framework [3]). A system view describes a certain facet of the system (e.g., structure, behavior or distribution). The use of diagram types provided by a modeling language is normative for specifying view mechanisms. However, diagram types (e.g., UML activity, class and state diagrams) [4][8] only provide separation of structure and behavior and do not inherently provide separation of crosscutting features. To better manage complexity an MDD framework should provide support for separating crosscutting features.

Aspect Oriented Software Development (AOSD) [9][10][11][12][13] supports horizontal separation of concerns by providing mechanisms for encapsulating crosscutting features using *aspects*. In our aspect oriented modeling (AOM) approach [20][21], crosscutting features are modeled as aspects and composed with the primary design model to form complete applications.

In this paper we present an aspect oriented model driven framework (AOMDF) that enables vertical and horizontal separation of concerns. The framework illustrates how aspect based techniques can facilitate the separation of concerns and ease the modeling and transformation design. Vertical separation of concerns is supported by providing techniques for transforming the models from one abstraction level to another. The models are transformed using mappings that are defined separately for the primary model and each of the aspects. Horizontal separation of concerns is realized by modeling crosscutting features separately as aspect.

Section 2 provides background information on AOM and model transformations. Section 3 describes the framework. Section 4 presents the framework using a scenario from a bank application. Section 5 discusses related work. Section 6 draws some conclusions and outlines planned work on the framework.

2 Background

2.1 Model Transformation

Many model transformation approaches are based on specifying mappings from source meta-model concepts to target meta-model concepts, as well as deriving target patterns based on source pattern recognition [26][27]. However, these meta-model mappings may not deliver the desired results. For example, 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 characteristics of the platform (e.g., deployment and distribution), it may

be necessary to transform instances of the same metamodel element differently. To derive a PSM, mechanisms provided in the platform as well as recommended patterns and practices should be utilized. For example, most middleware platforms provide specific services for handling security, persistence, and transactions. These services typically require specific protocols to be followed. Using a generic mapping of meta-concepts may not be appropriate when utilizing platform provided services and protocols. These pervasive features need to be treated explicitely to obtain the desired result. The AOMDF facilitate PIM to PSM mappings in which provided platform specific protocols are used.

MOF 2.0 Query View Transformation (QVT[6][7] is an ongoing standardization effort within the OMG. The aim of this process is to standardize a language for specification of model relations and transformations.

We base our mapping specifications on the current QVT submission. This has both drawbacks and benefits. A drawback is that the specification is a moving target and undergoing change. The specification also has some gaps and unfinished parts which makes it challenging to use. On the other hand the QVT will most likely be a standard and many industries are involved in its development. Also the joint submission specification is based on languages and tools already provided, e.g., [30][31]

Currently there are a variety of model transformation approaches and tools available. Quite a few of these are referenced in [27]. It is not clear to us how these approaches and tools will support the standardization effort in the model transformation area.

2.2 Aspect Oriented Modeling

There is ongoing research that investigates how to apply AOSD techniques at the model level [13][15][16][20][23]. The AOMDF is based on an AOM approach in which a design is expressed in terms of the following artifacts [20][21]:

1. A *primary model* that describes the business logic of the application.
2. A set of *generic aspect models*, where each model is a generic description of a crosscutting feature.
3. A set of bindings that determine 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 [20][21].

3 The Aspect Oriented Model Driven Framework

Figure 1 shows the major activities and artifacts supported in the AOMDF. The primary focus of the framework is the transformation of aspect oriented models from more abstract forms to more detailed forms. The major activities are partitioned into four categories: *source level*, *mappings*, *target level* and *model composition*. These activity categories are described below.

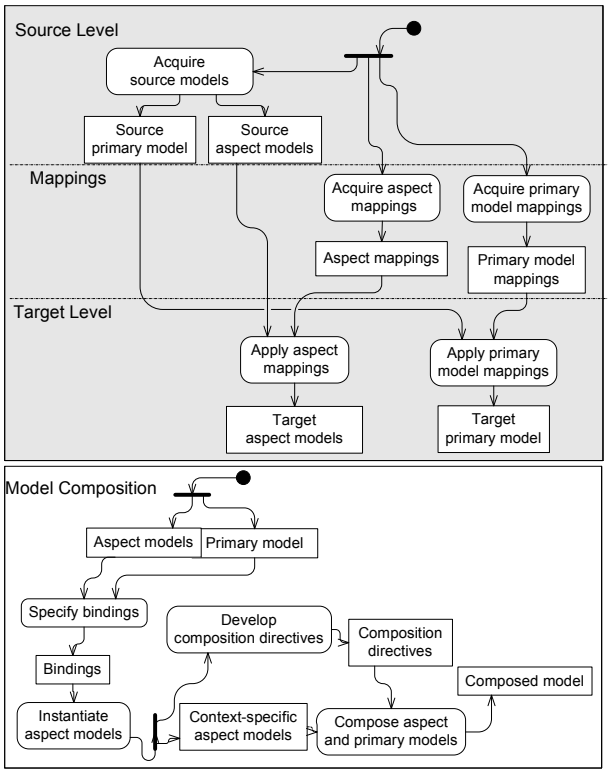

Figure 1 Aspect oriented model driven framework

The source level includes activities for acquiring or developing abstract aspect and primary models**.** At this level, the aspect models are acquired from an aspect repository if one is available or they are developed by the system architect. The primary model is developed by the system architect. The system architect decides what features will be included in the primary model and which will be treated as aspects. The decisions are based on functional and extra functional requirements. Extra functional requirements also called Quality of Service (QoS) requirements, such as security and transaction management are often pervasive. AOSD techniques are used to separate features that address these requirements from the primary business functionality.

The mappings category includes activities for developing or acquiring the corresponding target mappings for the aspect and primary models**.** The transformations between the source and the target levels

are defined by separate mappings for each aspect and the primary model.

The target level includes activities for applying the mappings to the source level primary and aspect models. The target detailed design models are obtained by applying the source to target transformations that are specified in the mappings.

The model composition part includes activities for instantiating and composing the aspect and primary models using bindings and composition directives [20][24]. Aspect models have to be instantiated before they can be composed. Instantiation is performed by binding the aspect model elements to the application specific model elements. Once the instantiation is done, the model composition is performed using the composition directives and a basic name matching procedure [20]).

The source and target levels have a recursive nature. Thus, the source level in one context may appear as the target level in another context. The source level and target level are relative to another.

AOMDF has two major variation points that must be fixed before the framework can be used. The two variation points are (1) the *framework levels*, and (2) the level(s) at which *composition* will be done.

Using the MDA terminology, the two main abstraction levels for models are PIM and PSM. The PIM and PSM are relative to the chosen platform (e.g., middleware platforms like J2EE, CORBA and .Net). Fixing the platform one may still define a set of source and target levels within the PIM and PSM context. For instance it may be desirable to perform transformations from PIM architecture model to a PIM detailed design model, and likewise to have several abstractions within the PSM level. Table 1 lists five different instantiation types of this generic framework based on different combination of the variations points.

	Instantiation types				
	1	2	3	4	5
Source level	pim	pim	pim	pim	psm
Target level	pim	psm	psm	psm	psm
Composition level(s)	pim	pim	psm	pim and psm	psm

Table 1: Potential framework instantiations

The composed model at the source level can be used for conformance checking of the composed model at the target level. However, conformance checking is beyond the scope of this paper.

The following is a list of the perceived benefits of the proposed framework:

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 source to target is less complex than the specification of the transformation of an integrated source model to target model, 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.
5. The aspects are often application independent (e.g., security and transaction). The aspect model and its mappings can therefore be reused across multiple applications and application domains once they are defined.

4 Illustrative example

We illustrate the framework with a distributed banking application that offers electronic money transfer using distributed transaction services.

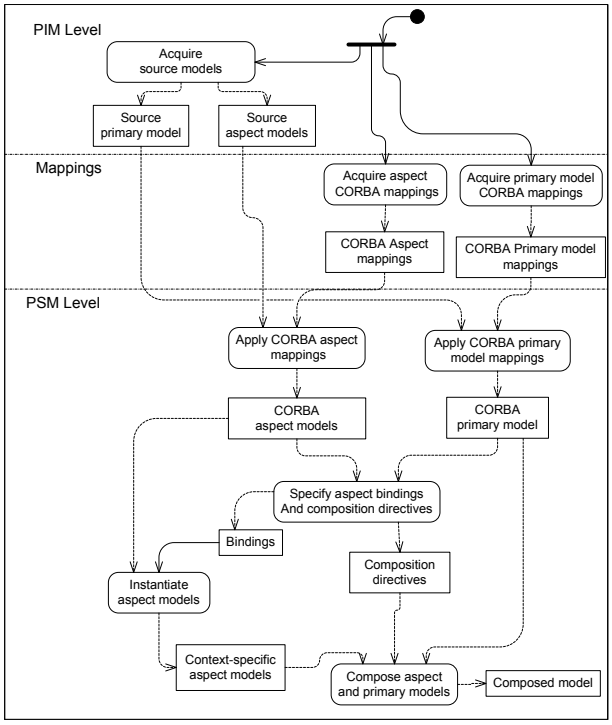

Figure 2 PIM to PSM framework instance

In the example, the framework is instantiated as follows:

- Source and target models are at PIM and PSM levels, respectively. The platform in question is CORBA

- The model composition is performed only at the PSM level.

Figure 2 shows the instantiated framework. Mappings are defined for a CORBA transaction aspect and money transfer scenario. They are applied on the PIMs to obtain the PSMs. The primary model is tagged to show where in the primary model the aspects are composed. Once the primary model is tagged, the composition is done as described in our previous work [20][21][24].

4.1 Acquire Source Models

We present a simple banking scenario and a transaction aspect as interaction diagrams to illustrate the instantiated framework.

4.1.1 Primary model

The bank consists of a set of accounts. The business functionality includes operations to open and close accounts. Withdrawal and deposit of specific amounts of money are provided for accomplishing money transfer. The transfer of money requires transaction control, which is modeled as an aspect. The money transfer scenario shown in Figure 3 is the primary model used in this example.

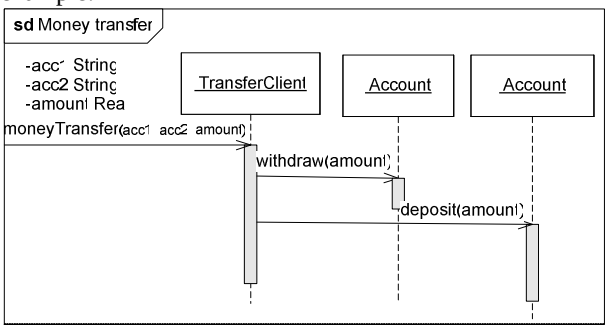

Figure 3 Banking scenario primary model

4.1.2 Transaction aspect

A transaction is an indivisible collection of operations between servers and clients that remains atomic even if some clients and servers fail. An atomic operation is an operation that is free of interference from concurrent operations performed by other threads in a system. Transactions are required to manifest the `ACID' properties [25]. While different middleware may provide different transaction models, a generic transaction model that captures the essence of distributed transactions can be specified at the PIM level. The generic model can then be transformed based on the specific protocol for each middleware.

Figure 4 shows a distributed transaction feature modeled as an aspect. The transaction aspect describes one-phase and two-phase commit distributed transaction protocols. The one-phase and two-phase commit protocols are as alternates in the figure.

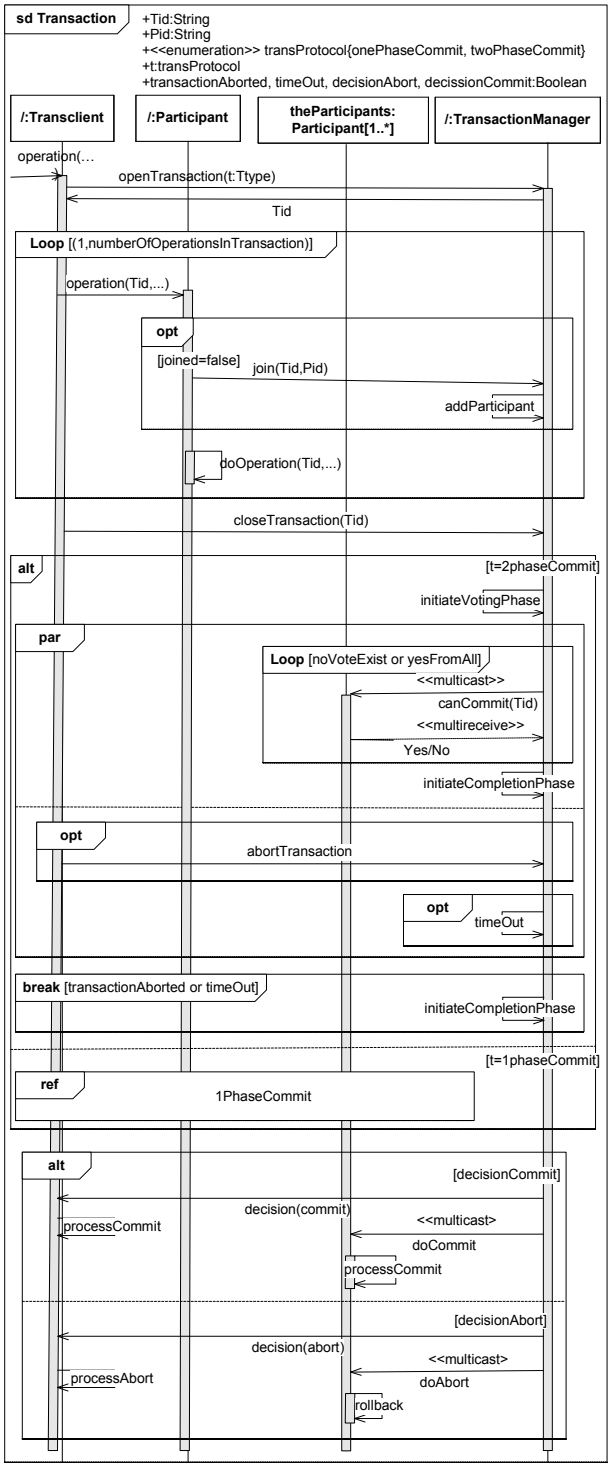

Figure 4 Transaction aspect

The transaction aspect has three main roles:

- A **Transaction Client** initiates the transaction and performs a collection of operations for the specific transaction.

- A **Participant** provides some service required by the Transaction Client. Figure 4 shows a collection of Participants representing the set of Participants involved in the transaction.
- A **Transaction Manager** is responsible for coordinating and managing transactions.

The Transaction Client initiates the transaction by sending the *openTransaction*. When the Transaction Manager receives *openTransaction* message, it opens a transaction and returns a transaction id (*Tid)*. This Tid is sent as a parameter in 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.

Two-Phase Commit Protocol: When the transaction client requests to close the transaction, the Transaction Manager starts the commit protocol according to the chosen transaction protocol type. The diagram in Figure 4 shows the details of 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 (*closing phase*), the Transaction Manager decides to abort or commit the transaction. The decision is multicast to all participants. At any time during the transaction, the transaction clients can request to abort the transaction or the transaction manager may timeout. Both requests result in the initiation of the completion phase. The Transaction Manager will then eventually decide to abort and all participants will be informed. Participants will then roll back the transaction individually.

4.2 Defining an Interaction Metamodel

QVT transformation specifications are metamodel based, and thus, to specify transformations, the source and target meta-models are needed. Both the source models in our example (the primary model and the transaction aspect) are specified using UML 2 interactions. The interactions metamodel is specified in the UML 2 standard[8]. However, the metamodel for interactions as specified in the UML 2 is fragmented, and the fragments are tied together via several other metamodel packages like the UML 2 kernel, the basic actions, and the basic behaviors. The mapping specifications would have been unnecessarily complex if we had used the UML 2 metamodel specifications directly. We have derived a simplified interaction metamodel including the basic concepts of interactions and their relationships. This model is shown in Figure 5.

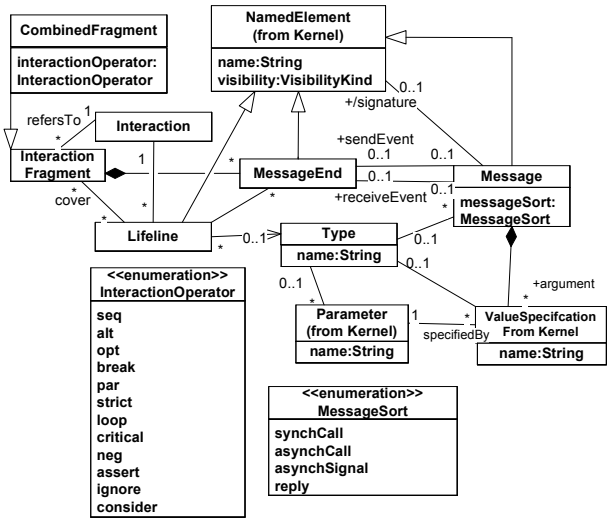

Figure 5 Simple interaction metamodel

4.3 Acquire Primary Model Mapping

One possible CORBA mapping for the primary model is to derive a PSM sequence diagram showing the CORBA object interactions. Stereotypes can be used to indicate the kind of CORBA objects. This is a straightforward mapping where CORBA stereotypes are added and primitive types are converted if they are different. The result is shown in Figure 6.

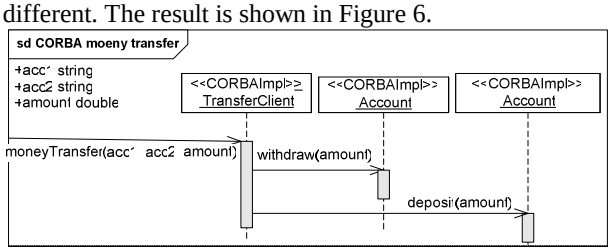

Figure 6 PSM sequence diagram

Another mapping, is to derive an IDL representation based on the specified source model. From this, stubs, skeletons and helper classes can be generated using an IDL compiler. A QVT specification for mapping interaction diagrams to CORBA IDL is shown in Figure 7. The UML profile for CORBA [37] is used as the target metamodel. This representation is compliant with an IDL representation and may serve as the source for an IDL compiler.

Two mappings are defined in Figure 7. The upper mapping derive the CORBA interfaces with operations, the lower add directed associations. The left hand side describes a pattern that should be matched in order for the mapping to execute. The pattern is an instantiation of the interaction metamodel. The header of the package specifies input and output (*Lifeline* and *CORBAInterface*, respectively). These are the anchors of the structures of the left hand side and right hand side respectively. A

Lifeline has a set of zero or more receive MessageEnds, Sets are indicated with the multiplicity star. According to the interaction metamodel *Lifeline*, *Type*, *Message* and *MessageEnd* have names. These are not explicitly shown in the source patterns, but are used to derive the target structures.

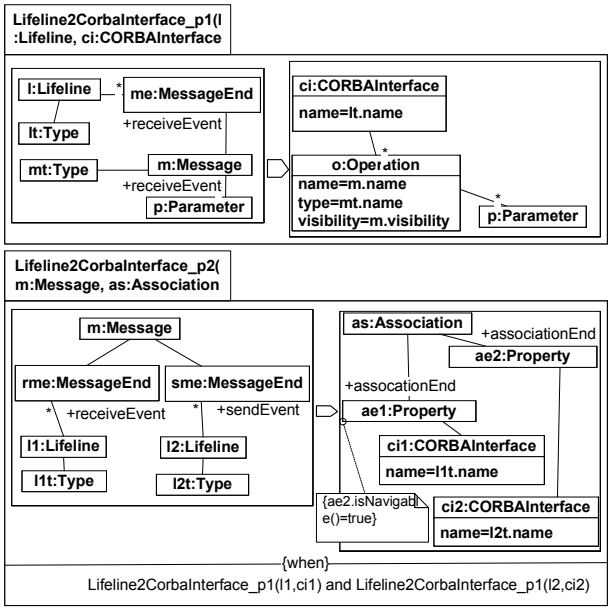

Figure 7 QVT primary model mapping specification

The mappings produce a *CORBAInterface* for each lifeline type having the same name as the lifeline type name. For every receive MessagEnd a corresponding operation is added. The parameter specifications remain the same in both source and target. This assumes that the primitive types of source and target are equal, else a type mapping would be needed. The different members of the patterns are referred using their names. According to the QVT specification the names are also used to decide whether to create new elements or edit existing ones. For example if the *CORBAInterface* already exists, only new operations are added.

An imperative pseudo code specification for this mapping is as follows:

create *Transclient* Interface;

add *moneyTransfer(..)* **to *Transclient*** Interface;

create *Account* Interface;

add *withdraw(amount)* **to *Account*** Interface;

add *deposit(amount)* **to *Account*** Interface;

add directed association between **Transclient** and **Account**

The resulting CORBA specification is shown in Figure 15

4.4 Acquire Aspect Mapping

When developing the aspect mapping we want to utilize the transaction service provided by CORBA. The mappings to transform the PIM transaction aspect shown in Figure 4 must include all interactions that involve the *TransactionManager* and all transactional interactions between *Participants* and *Transclient*. These are grouped into six sets of mappings as follows:

1. The *openTransaction* message Transclient to TransactionManager.
2. The *join* message from Participant to TransactionManager.
3. The *closeTransaction* message Transclient to TransactionManager.
4. The *abortTransaction* message Transclient to TransactionManager.
5. The *canCommit* message from TransactionManager to Participants.
6. Other mappings involving decisionCommit(commit), decisionAbort(abort), doCommit and doAbort.

We describe the aspect mappings first using an imperative style and we then give examples of how they can be expresses using QVT.

// *openTransaction* pseudo code mapping specification:

replace TransactionManager **by** {ORB; CurrentHelper; Current*;}*

replace *openTransaction* message **from** *Transclient* **to** *TransactionManager* **by {**

resolve_initial_references("TransactionCurrent") **from** *Transclient* to *ORB*;

narrow(..) **from** *Transclient* **to** *CurrentHelper;*

}

add *set_timeout(time)* message **from** *Transclient* **to** *Current*;

add *begin()* message **from** *Transclient* **to** *Current;*

// *join* pseudo code mapping specification:

replace *TransactionManager* by {*Control; Coordinator;}*

replace join(Tid, Pid) message *from* Participant *to* TransactionManager **by {**

get_control **from** Participant **to** Current;

get_coordinator **from** Participant **to** Control;

*register*_resource(Pid) **from** Participant **to** Coordinator*;*

}

delete add_participant message **from** TransactionManager **to** TransactionManager*;*

// closeTransaction pseudo code mapping specification:

replace ***closeTransaction*** message **from** *Transclient* **to** *TransactionManager* **by** *commit()* from *transClient* to *Current.*

***// abortTransaction* pseudo code mapping specification:**

replace *abortTransaction* message **from** *Transclient* **to** *TransactionManager* **by** *rollback()* from *Transclient* to *Current.*

***// canCommit* pseudo code mapping specification:**

replace *canCommit* message **from** *TransactionManager* **to** *Participants* **by** *prepare()* from *Current* to *Participants*

.

***// Other* pseudo code mapping specification:**

delete *decision(commit)* message **from** *TransactionManager* **to** *Transclient.*

delete ***decision(abort)*** message **from** *TransactionManager* **to** *Transclient.*

replace *doCommit* message **from** *TransactionManager* **to** *Participants* **by** *commit()* from *Current* to *Participants..*

replace *doAbort* message **from** *TransactionManager* **to** *Participants* **by** *rollback()* from *Current* to *Participants..*

The initiateVotingPhase and initiateCompletionPhase messages have no CORBA equivalents. They are retained in the model to provide logistical information to developers, however, no mappings are applied to them.

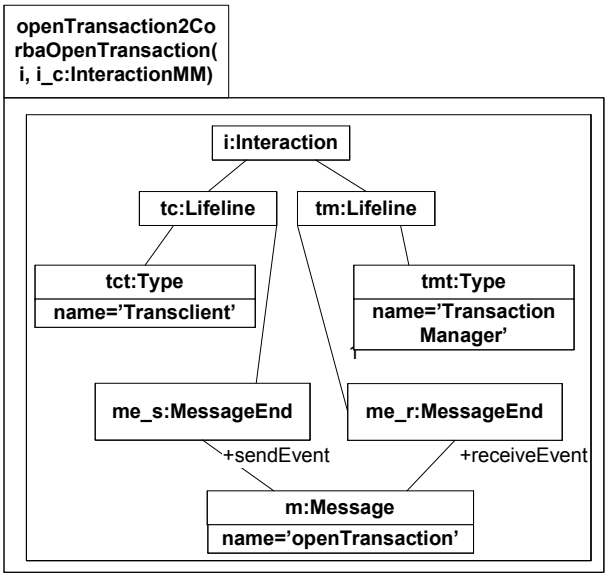

Figure 8 source part of the QVT aspect model mapping for the "openTransaction" operation

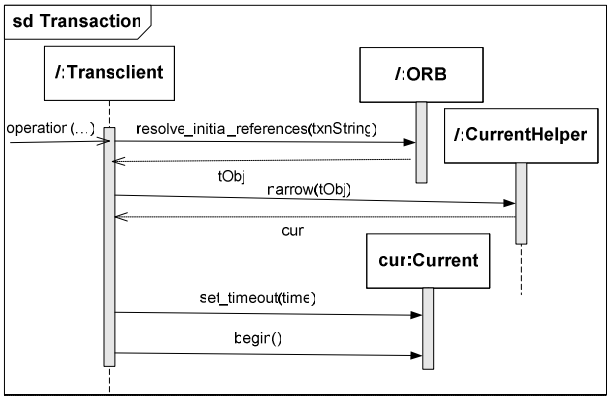

Figure 9 Open transaction CORBA counterpart

Figure 8 shows the source part of the mapping for the open transaction. The pattern defined in the figure is basically to recognize the *openTransaction* message between the *Transclient* and the *TransactionManager*

The derived target of the open transaction is shown in Figure 9.

The mapping used to derive the CORBA target model is shown in Figure **10**. The target model is obtained in three steps. The first two steps produce the operation/return message pairs; *resolve_initial_references(txnString)* and *narrow(tObj)*. The mapping specification for these are shown in the upper part of Figure **10**. The specification of the *set_timeout(time)* and the *begin()* operations are shown in the lower part of the figure.

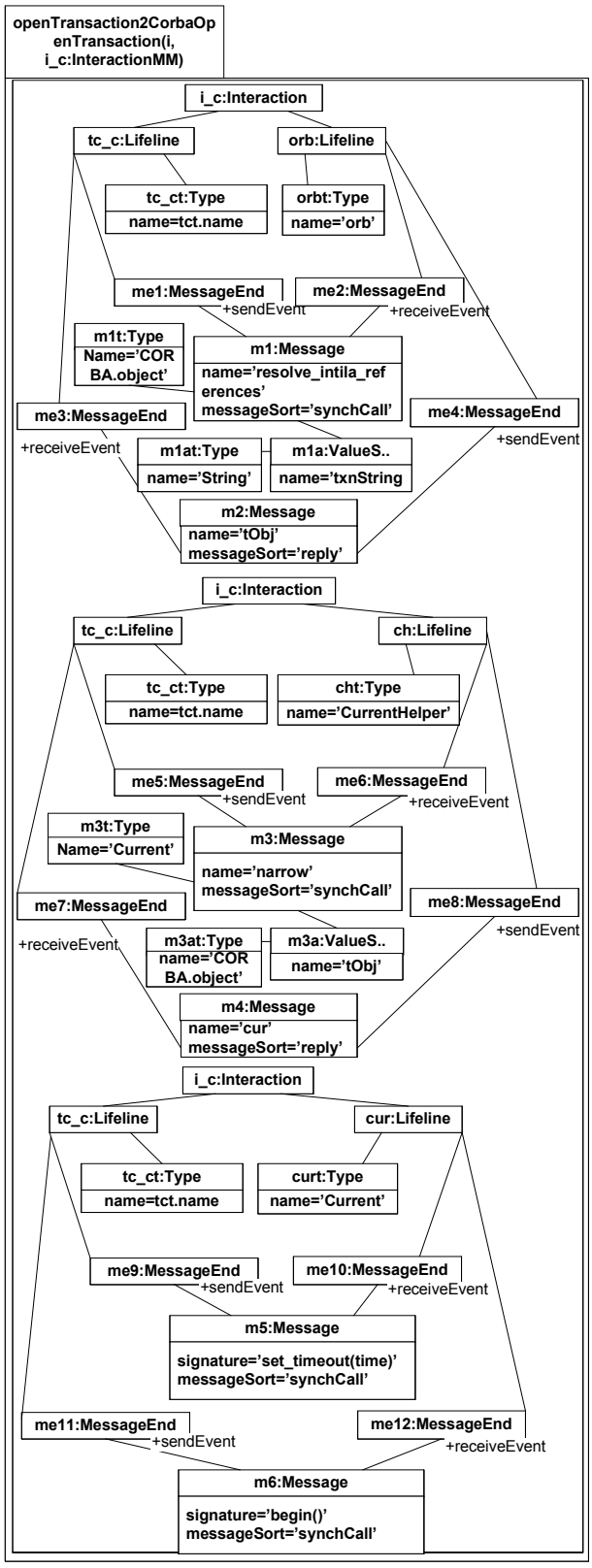

Figure 10 **Target part of the *openTransaction* mapping**

Figure 11 shows the source part of the QVT mapping for the *join* message.

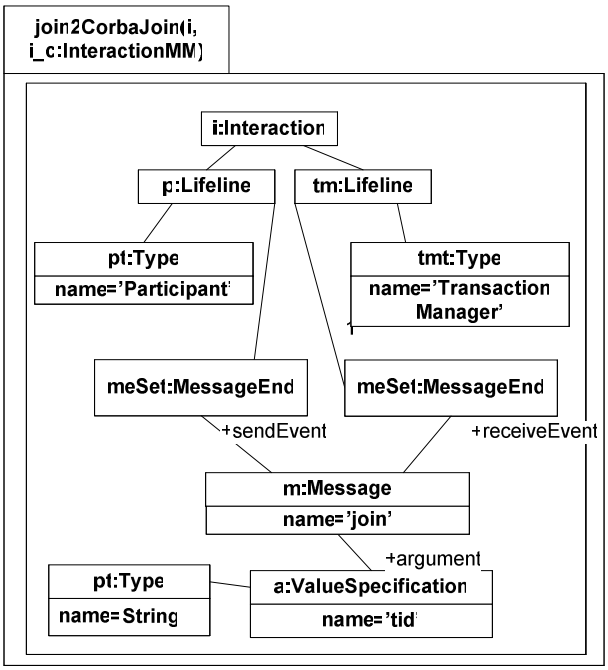

Figure 11 Source part of the QVT aspect model mapping for the "join" operation

The pattern defined in Figure 11 essentially recognizes the *join* message between the Participant and the TransactionManager.

The derived target of the *join* transaction is shown in Figure 12.

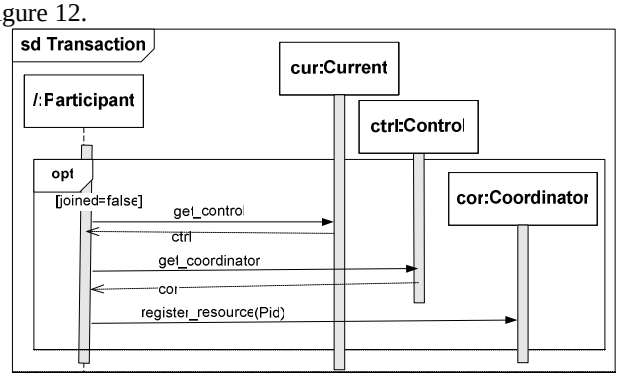

Figure 12 join transaction CORBA counterpart

The corresponding mapping specification for the CORBA target Model is shown in Figure 13

As the example illustrates the mapping specifications of both the open transaction and the join are complex. This is because these specific messages need to be treated explicitly in order to utilize the CORBA transaction service and follow the required protocols.

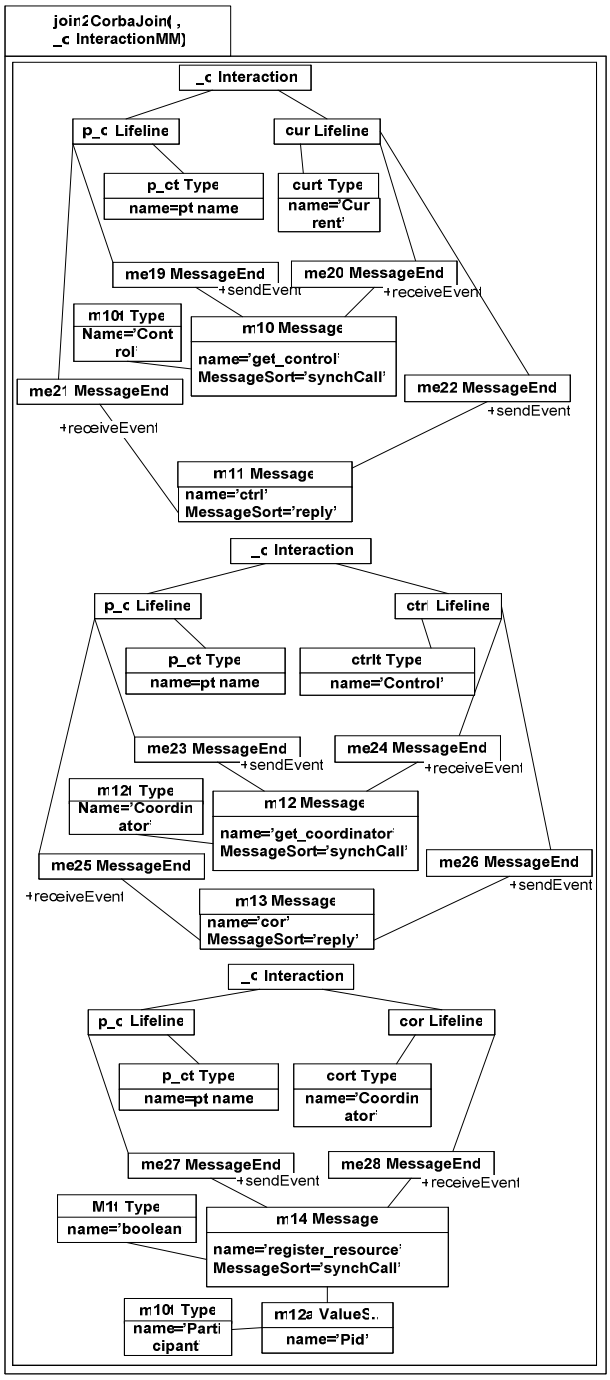

Figure 13 Target part of the QVT aspect model mapping for the "join" operation

However, since transactions are application independent, the mapping specification is highly reusable. In the example there is a repeating pattern that is used in order to specify the derivation of the target. It may be possible to obtain more powerful mapping specifications through parameterized patterns.

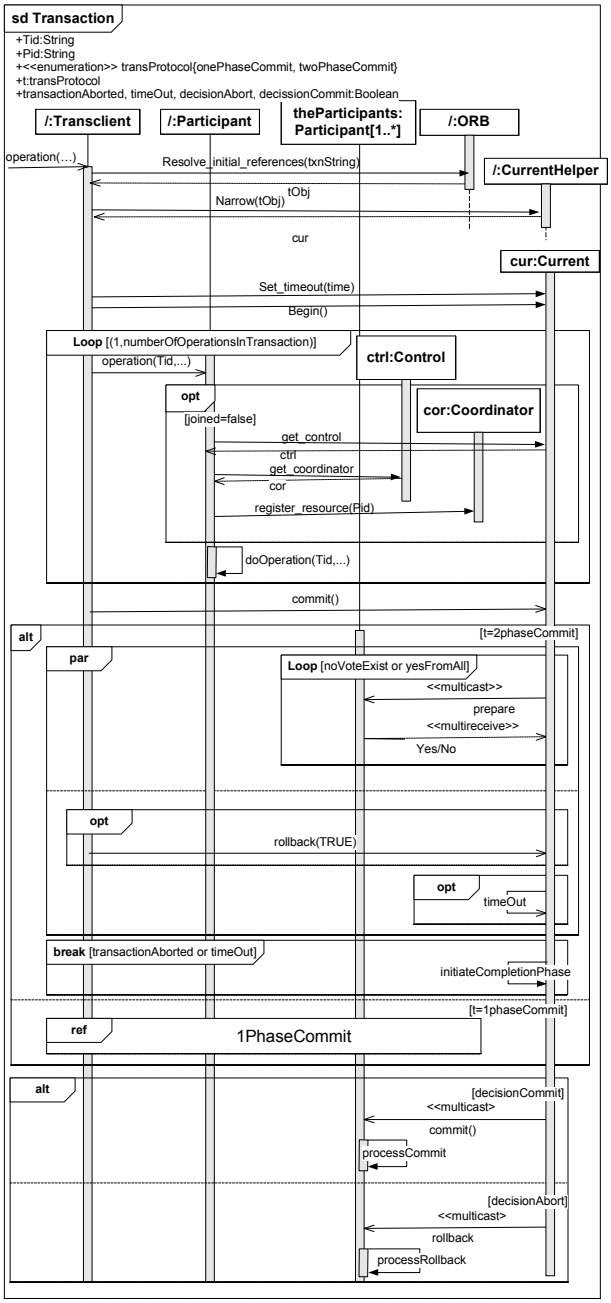

Figure 14 CORBA transaction PSM

4.5 Apply Mapping

Figure 15 shows the CORBA IDL interface generated by applying the PIM to PSM mappings specified in section 4.2 to the primary model. Figure 17 presents the composed sequence diagram that results from applying all the aspect mappings to the PIM transaction aspect shown in Figure 4.

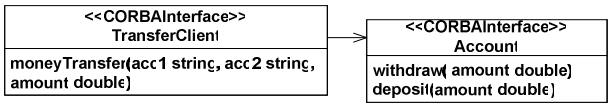

Figure 15 Generated Interfaces based on UML profile for CORBA

4.6 Specify Aspect Bindings & Instantiate Aspect Models

Before composition, the primary model is tagged to define where in the primary model the aspects are composed. The aspect tagging is based on AOP waving mechanisms. Figure 16 shows the banking scenario and how the lookup aspect and the transaction aspect should be weaved into the model. The lookup aspect is another aspect that be defined similar to the transaction aspect. The <<aspect>> stereotype is used to model aspect tags. Subsequent to the incoming *moneyTransfer* method call the lookup aspect is performed twice to get the handle of the accounts involved. The transaction aspect is a stereotyped combined fragment that encompasses the transactional method calls. Combined fragments are constructs defined for interaction diagrams in UML 2.

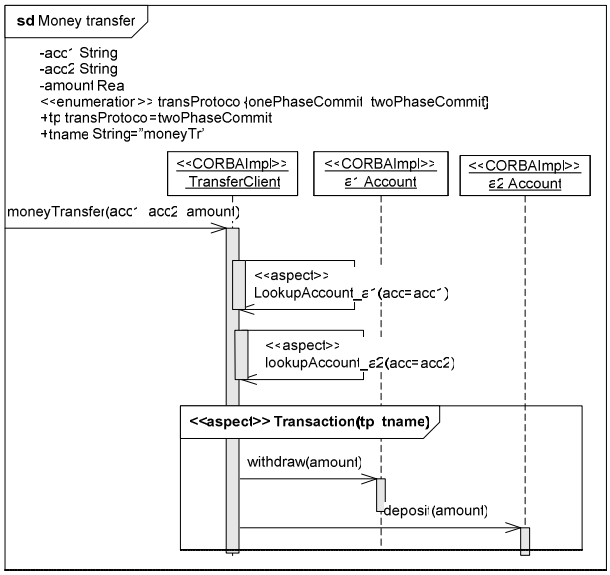

Figure 16 Primary model tagged with aspects

Once, the primary model is tagged with the aspects, the aspects and primary model are composed using the bindings and composition directives to obtain an integrated design view referred to as the composed model [20]. This is shown in Figure 17.

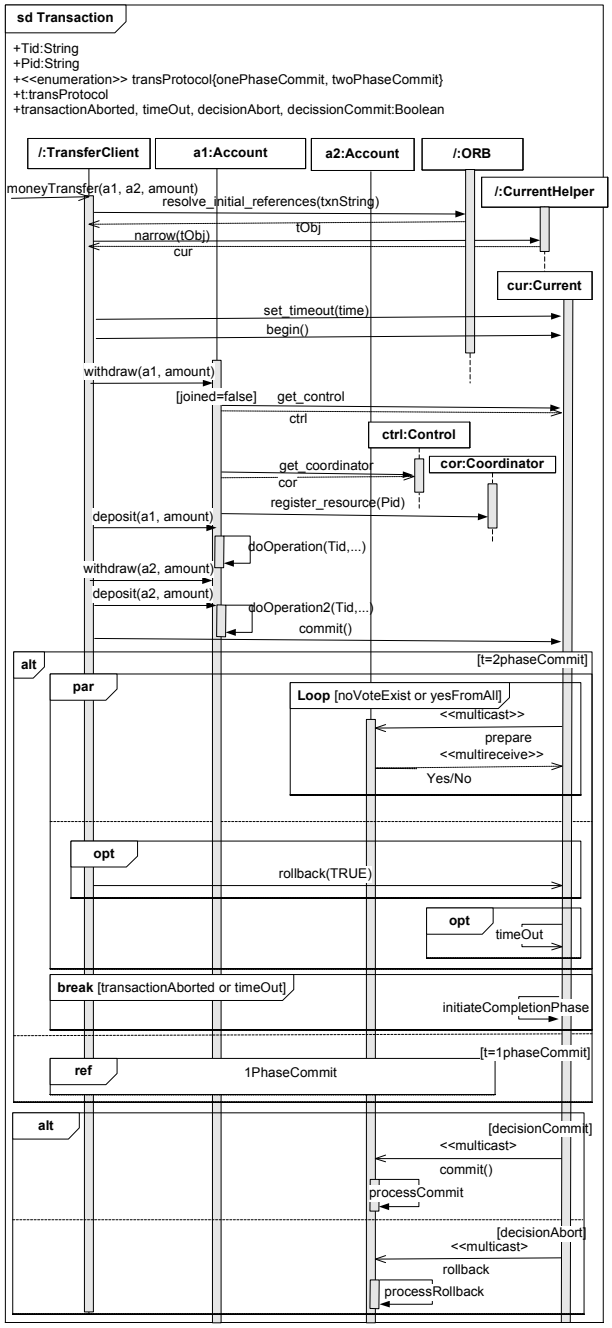

Figure 17 PSM Composed Model

5 Related work

Several researchers has done work on developing transformation languages and tools. ArcStyler [32], EXMOF [29], Objecteering [33], and Tarzan/XMorph [34] are some of some transformation engines available. TopMdl [35] is an international open-source initiative launched to provide an extensible framework for model-driven experimentation. Most of the tools/languages are

domain-specific and are either imperative or declarative. The proposed framework shown in the paper uses both declarative and imperative languages for transformations and hence can be used in wider scope.

Jacobson [17][18] 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 [5]. The work does not explicitly give details about transformation of models, rules of composition, structural relations, etc.

Reina et al. [16] 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. [23], 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. [19] 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 [22].

6 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). The AOMDF provides additional support for specifying transformations. The AOMDF allows us to separate out the mapping specification for pervasive features from the mapping specification of the primary model. The aspect mapping specification then becomes reusable and the mapping specification of the primary model becomes simpler

The paper illustrates the transformation of a platform independent distributed transaction aspect to a platform specific transaction aspect. We also describe the integration of the transaction aspect in the context of a net banking application. The example illustrates that the mapping of pervasive services can be complex, for instance since we need to obtain specific mappings of specific operations.

Currently we are working on techniques to resolve conflicts that can occur if more than one aspect is composed with the primary model. Verifiable composition techniques that discharge proof obligations during composition are being developed.

In the future, we 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.

References

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

[2] Soley, R.M, Frankel, D.S.,Mukerji, J.,Castain, E.H., Model Driven Architecture - The Architecture Of Choice For A Changing World, OMG 2001. http://www.omg.org/mda/Soley,

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

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

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

[6] MOF™2.0 Query/View/Transformation RFP, OMG Document:ad/2002-04-10

[7] 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

[8] UML™ 2.0, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML

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

[10] 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.

[11] H. Ossher and P. Tarr. Using multidimensional separation of concerns to (re)shape evolving software. Commun. ACM, 44(10):43ñ50, 2001.

[12] J. Kienzle and R. Guerraoui. Aop: Does it make sense? the case of concurrency and failures. In Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP, pages 37-61. Springer-Verlag, 2002.

[13] 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.

[14] 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.

[15] R. Silaghi, F. Fondement, and A. Strohmeier. Towards an MDA-Oriented UML Profile for Distribution. In Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference, EDOC, Monterey, CA, USA, September 2004.

[16] 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.

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

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

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

[20] 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.

[21] 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.

[22] E. Song, R. Reddy, R. France, I. Ray, G. Georg, R. Alexander. Verifying Access Control Properties using Aspect Oriented Modeling. Accepted in 10th ACM Symposium on Access Control Models and Technologies (SACMAT), Scandic Hasselbacken, Stockholm, June 1-3, 2005.

[23] 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.

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

[25] George Coulouris, Jean Dollimore and Tim Kindberg. Distributed Systems Concepts and Design (3rd Ed.) Page 471. International Computer Science Series, Addison-Wesley/Pearson Education, USA, 2001.

[26] Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10), QVT-Merge Group 1.8, OMG document ad/2004-10-04. www.omg.org

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

[28] R. Hubert. Convergent Architecture: Building Model Driven J2EE Systems with UML. John Wiley & Sons

[29] EXMOF - Queries, Views and Transformations on Models using MOF, OCL and EXMOF, Joint 2nd revised submission. Compuware Corp., SUN Microsystems. ad/2004-10-03

[30] Tata MasterCraft http://www.tata-mastercraft.com/index1.asp

[31] Atlas Transformatino Language, www.tni-software.com/?p=mda

[32] ArcStyler http://www.io-software.com/products/arcstyler_overview.jsp

[33] Objecteering/Introduction User Guide. version 5.3 - CODOBJ 001/001. www.objecteering.com

[34] K. Duddy, A. Gerber, M.J. Lawley, K. Raymond, J. Steel. Model Transformation: A Declarative, Reusable Patterns Approach. In Proceedings 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003), pp 174-185

[35] P.A. Muller, P. Studer, J.M Jézéquel. Model-driven generative approach for concrete syntax composition In workshop on Best Practices for MDSD (OOPSLA'2004), October 2004

[36] D. M. Simmonds, S. Ghosh and R. B. France." Middleware Transparent Software Development & The MDA," In Proceedings of the WiSME Workshop in Software Model Engineering at UML'2003, San Francisco, October 2003.

[37] UML™ Profile for CORBA™ version 1.0, April 2002, formal/02-04-01.