SUPPORTING ASPECTS IN MDA

Vinay Kulkarni and Sreedhar Reddy
Tata Research Development and Design Centre,
54 B, Hadapsar Industrial estate, Hadapsar,
Pune, 411 013, INDIA
{vinayk, sreedharr}@pune.tcs.co.in

Abstract

For developing large and complex applications, industrial practice uses a combination of non-formal notations and methods. Different notations are used to specify the properties of different aspects of an application and these specifications are transformed into their corresponding implementations through the steps of a development process. The development process relies heavily on manual verification to ensure the different pieces integrate into a consistent whole. This is an expensive and error prone process. Model-driven development approach addresses this problem by providing a set of modeling notations for specifying the different layers of a software architecture and a set of code generators for transforming the models into an implementation. Model-driven development has resulted in improved productivity, better quality and platform independence. However, it has not been very successful in supporting reuse and system evolution due to inadequate modeling support for clear separation of concerns and their composition. The model driven architecture (MDA) initiative of OMG aims to shift the focus of software development further towards modeling. With model content in system specifications increasing more and more, it is critical to address the issue of separation of concerns at model level. Aspect oriented programming addresses separation of concerns at the code level. In this paper, we argue for supporting aspects in MDA. We propose an approach wherein different aspects can be specified using different modeling notations and propose model transformations as a mechanism for weaving them. We discuss several issues including tool support that need to be investigated in order to support multi-dimensional separation of concerns in MDA.

Keywords

MDA, Aspect oriented programming, Model driven development, Separation of concerns, Model transformations, Meta modeling

1. Introduction

Model-driven development has resulted in improved productivity, better quality and platform independence [14, 8]. However, it has not been very successful in supporting reuse and system evolution due to inadequate modeling support for clear separation of concerns and their composition. To facilitate traceability, reuse and evolution, a system needs to be specified as a composition of multiple views corresponding to multiple stakeholders and their concerns [11]. Aspect oriented programming (AOP) addresses separation of concerns at the code level wherein both *aspects* and *components* are typically specified in object oriented paradigm [3]. Aspects are woven into components using a set of pre-defined composition relationships at specified *join points*. The model driven architecture (MDA) initiative of OMG aims to shift the focus of software development further towards modeling [9]. With model content in system specifications increasing more and more, it is critical to address the issue of separation of concerns at model level [15]. In this paper, we present an approach to specify aspects as models and their weaving as model transformation. We also discuss various issues that need to be investigated and the required tool support.

Section 2 makes a case for supporting aspects in MDA. Section 3 describes the proposed approach. Section 4 discusses open issues and the tool support required. Section 5 discusses related work.

2. Case for aspects in MDA

The MDA defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. To this end, the MDA defines an architecture for models that provides a set of guidelines for structuring specifications expressed as models. A system is typically specified in three kinds of modeling layers namely *Computation independent model* (CIM), *Platform independent model* (PIM) and *Platform specific model* (PSM). A CIM specifies the system in terms of end-user or domain concepts and can be seen as requirements specification of the system. A PIM specifies computational realization of the system in terms of platform-independent abstractions. A PIM can be seen as a refinement of a CIM just as a design is a refinement of a requirements specification. A PSM specifies the implementation of the system in terms of a set of platform-specific abstractions. A PSM can be seen as a realization of a PIM just as code is a realization of a design. The refinement relationships between various kinds of models are specified as a set of mappings. A mapping specification can be used to derive, wherever possible automatically, one kind of a model from the other. Specifying a CIM is largely a manual activity. A PIM is derived partly in an automated manner from a CIM and refined further manually. A PSM is largely derived automatically from a PIM.

Aspect-orientation advocates specification of a system as a (de)composition of concerns along several dimensions of interest leading to clear traceability across different levels of system specification namely requirements, design and code.

A system has multiple stakeholders each having own view of the system that is typically captured in a model form. A CIM captures the requirements of a system in terms of these multiple viewpoints. Each viewpoint can be seen as a dimension of concern as advocated in aspect-orientation. Organization of a CIM as a set of concerns / viewpoints enables concurrent development, ease of change management and evolution through additive changes. Since a PIM too is largely specified manually, preserving this organization at PIM level will provide the same benefits. Lack of separation of concerns at PIM level will result in entangled and scattered models that are difficult to comprehend and trace. In MDA, parts of PIMs are generated from CIMs and parts are specified manually. This part-generated-part-manual nature of a PIM exacerbates the problems of traceability and comprehension.

In MDA, a PSM is automatically derived from a PIM through a set of transformation specifications. Having separation of concerns at PIM level will enable specification of PIM-to-PSM transformation to be decomposed into a set of concern specific transformations. Such a composition architecture will enable plug-and-play, change isolation, ease of evolution through additive changes to PIM-to-PSM transformation.

3. An approach for supporting Aspects in MDA

Specification of a typical business system caters to functional and several non-functional requirements like concurrency management, performance, security etc. In traditional AOP parlance, functional requirements can be viewed as *components* and non-functional requirements as *aspects*. In the MDA approach, the component and the aspects will be specified in their own modeling languages. The weaving of an aspect into the component can be seen as transformation of the component model as shown in Figure 1. The model transformer takes the component model and the aspect model as input to produce a transformed component model. The transformer itself is specified as a model [10].

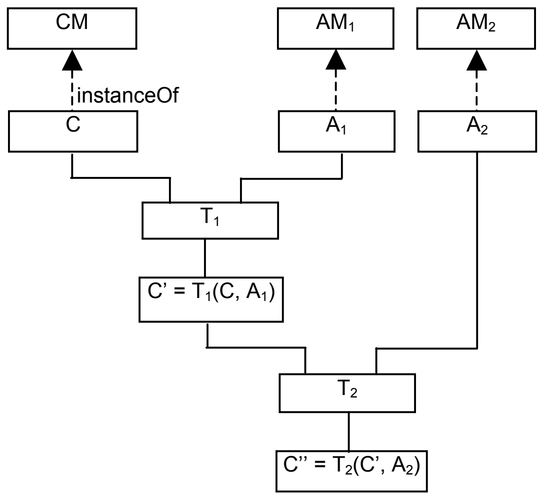

Figure 1. Aspect weaving as model transformation

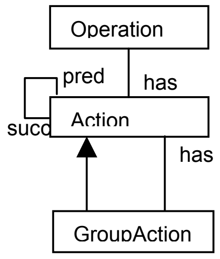

Figure 2. Action model

Consider a business application in which the business functionality is specified in terms of class models. For the purpose of this discussion we view operations as being specified in terms of a simplified action model as shown in Figure 2.

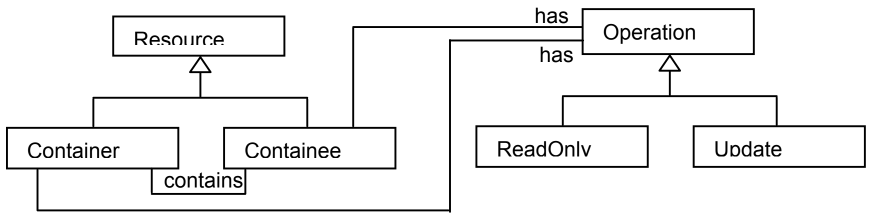

Figure 3. Concurrency management aspect meta model

Consider the concurrency management aspect. Here, we view the system as comprising of *Containers* and *Containees* having two kinds of *Operations* namely *ReadOnly* and *Update*. When a container reads its contents, they should be consistent for the duration of the read operation. This is typically achieved by taking a *ReadLock* on the contents being read. When a containee is being updated, an *UpdateLock* is taken to prevent concurrent read / update of the same. Figure 3 describes a model for the concurrency management aspect.

The concurrency management meta model can be specified as a view over the functionality meta model as shown in Figure 4.

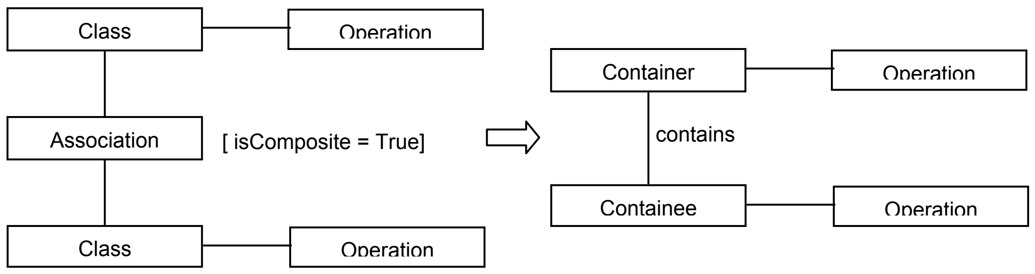

Figure 4. Concurrency management meta model as a view over the functionality meta model

The weaving of the concurrency management aspect into the functionality can be seen as a transformation as shown in Figure 5. The transformation is specified in terms of model patterns wherein each occurrence of the left hand side pattern is replaced by an occurrence of the right hand side pattern. Figure 5 depicts a read operation being transformed to take a *ReadLock* before performing the read action.

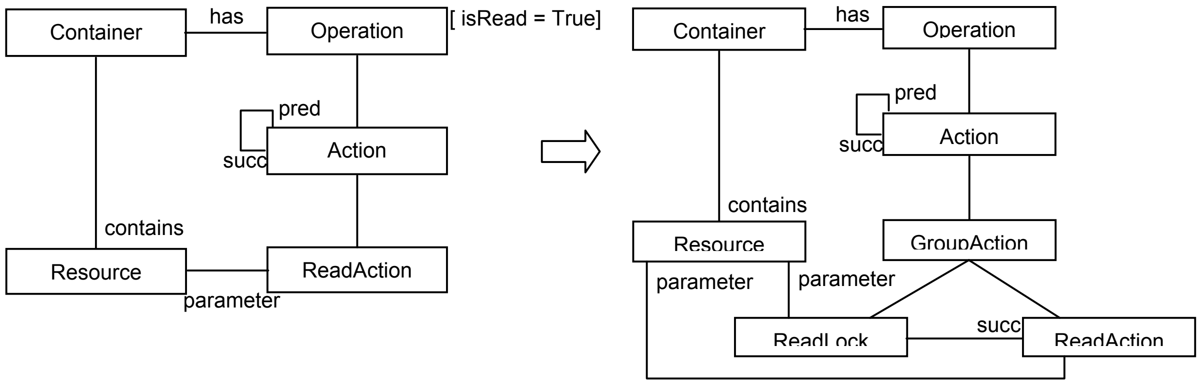

Figure 5. Weaving of the concurrency management aspect

Consider the performance aspect of a business application. In a transaction, multiple updates to the database of an object can be replaced by a sequence of in-memory update operations followed by a single (the last) database update operation. Similarly, multiple reads of an object from the database can be replaced by a single (the first) database read operation followed by a series of in-memory read operations. The read and update operations can be interspersed. A transaction may be composed of several operations each developed independently of others. An operation may read an object from the database and may write the same to the database. The first step of transformation is to flatten the transaction into a sequence of, possibly interspersed, database read and write operations. The performance optimisation transformation is applied over this view.

The performance aspect meta model can be specified as a view over the functionality meta model as shown in Figure 6.

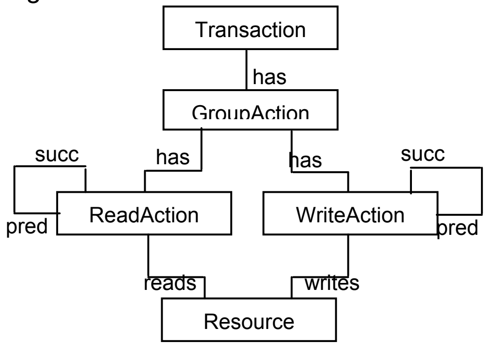

Figure 6. Performance aspect meta model

The weaving of the performance aspect into functionality can be seen as a transformation as shown in Figure 7. Figure 7(a) depicts the transformation that transforms the first read action to the database read action. Figure 7(b) depicts the transformation that transforms all subsequent database read actions to in-memory read action. Figure 7(c) depicts the transformation that transforms the last write action to the database write action. Figure 7(d) depicts the transformation that removes all previous database write actions to in-memory write actions.

Consider the workflow aspect. The system is viewed as a set of actions performed on a set of resources by a set of roles. A user plays a role on a resource. An action can only be performed by a user if it belongs to the role the user is playing on the resource. The workflow aspect model is specified in terms of a meta model shown in Figure 8.

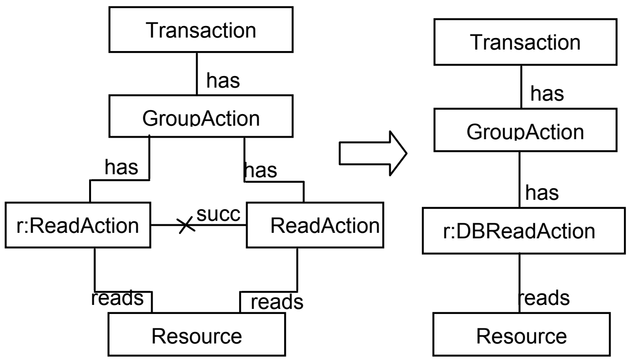

Figure 7(a). Transformation of the first read action

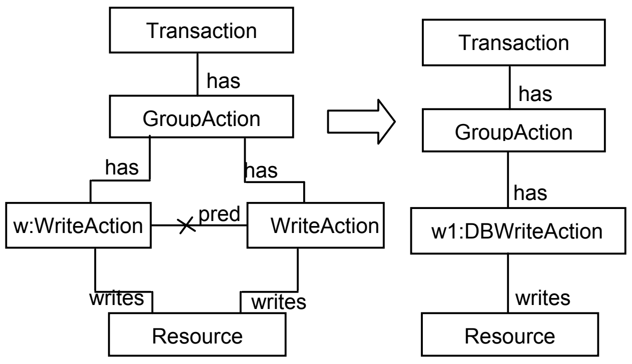

Figure 7(c). Transformation of the last write action

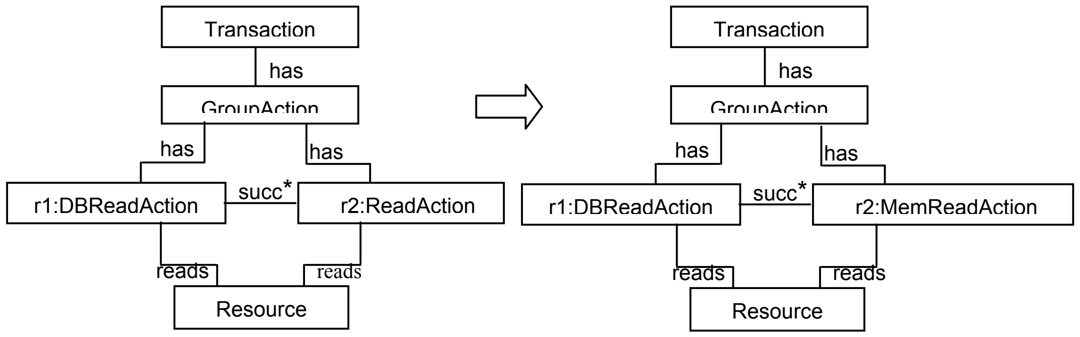

Figure 7(b). Transformation of other read actions

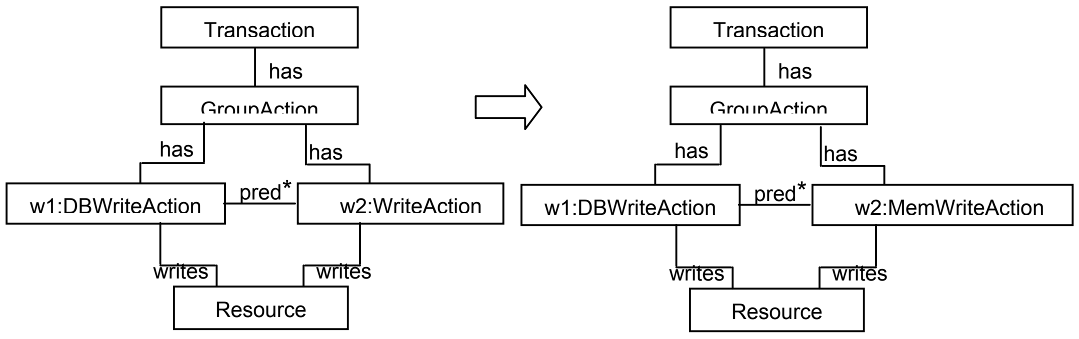

Figure 7(d). Transformation of other write actions

The weaving of the workflow aspect into the functionality can be seen as a transformation as shown in Figure 9. The transformation introduces a check for verifying if the current user is authorised to perform the action on the resource.

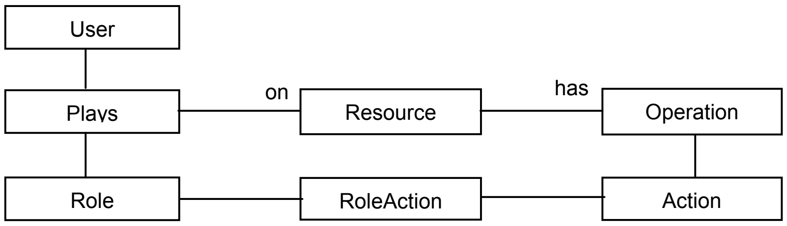

Figure 8. Workflow aspect meta model

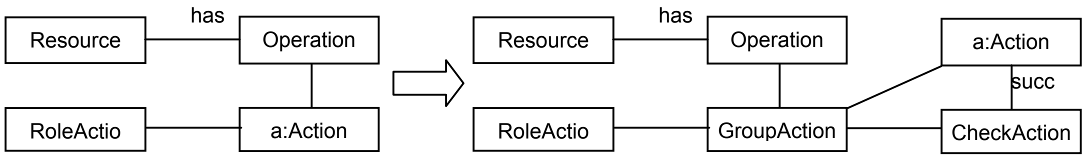

Figure 9. Weaving of the workflow aspect

4. Discussion

It is not clear which facets of a system deserve to be treated as aspects. There is a need to identify which of these aspects need to be separately specified. For instance, it is not clear how to cleanly separate the performance aspect from functionality. There is a need to investigate how these aspects can be modelled and what the right kind of abstractions for modeling them are to satisfy the various 'ities' like maintainability, reusability etc. For instance, how does one model a design for better maintainability?

Aspects may overlap each other. This may introduce a dependency on the order of their weaving. In such cases, how does one ensure that properties of all aspects hold after their weaving?

Some of the aspects, for instance some design patterns, may not be amenable to be specified completely in a model form that can be transformed into an implementation. Some of these may have to be directly implemented in a programming language. This gives rise to the issue of how one integrates these 'black box implementations' into the aspect modeling framework. For example, how does one weave other modelled aspects into these 'black box implementations'?

An aspect specification may exist partly in model form and partly in code form. What's the right approach to integrate such aspects into the aspect modeling framework?

A system is organized as a set of independently specified aspects. The knowledge of weaving an aspect is hidden inside the transformation. This gives rise to the issue of traceability from an aspect to the final implementation. It is not clear how to compute the impact of a change in an aspect on the final implementation of the system. This information would be critical for 'what if analysis', estimating testing efforts, managing releases etc.

There is a need to investigate and devise suitable processes that aid in arriving at various aspect models from system requirements. There is a need for well-defined guidelines and best practices to enable a non-expert developer make the best use of the proposed approach.

Supporting aspects in MDA raises several tooling requirements. The modeling tool should be extensible to support new modeling languages. This is required to define new aspect models and relate them to the existing component models through model transformation mechanism. The model transformation tool should have adequate support for pattern matching and composition. It should provide support for incremental reconciliation of models. The performance of the tool should scale up to cater to the demands of enterprise class applications.

There should be tool support for intelligent debugging at aspect model level. This is significant because aspects are specified independent of each other and are woven together into the final implementation code. A bug detected at code level should be traceable back to the aspect specification.

There should be support, preferably tool-aided, for aspect-based testing. Since aspects are independently specified, it should be possible to specify test cases for an aspect independently and compose the test cases to arrive at the system level test cases.

There is a need for tool support to check the consistency of aspect composition. For instance, when two aspects overlap it should be possible to check if it is safe to put them together so that the respective properties of each aspect hold in the woven model.

5. Related work

Robert France et al discuss the need for investigating which concerns are amenable for specification as aspects and the need for suitable techniques for their composition [12].

Siobhan Clarke et al propose a technique for designing reusable aspects using *composition patterns* [13]. We feel this technique cannot address complex composition scenarios that require model reorganisation. The power of model transformation, as proposed in this paper, is required to address complex composition requirements.

Several techniques have been proposed for modeling aspects in UML [1, 5, 6, 7]. Essentially, they provide an abstract syntax for aspect-oriented programming languages like AspectJ [2], Hyper/J [4] etc. and a visual notation for the same. They do not discuss what the right kind of abstractions are to represent various aspects. The weaving techniques proposed therein are not adequate to address complex composition scenarios requiring model transformations.

6. Summary

In this paper, we argued for supporting aspects in MDA. We proposed a model transformations based approach for aspect weaving. We discussed several issues that need to be investigated to support multi-dimensional separation of concerns in MDA. We also discussed some tooling issues.

References

1. Dominik Stein, Stefan Hanenberg, and Rainer Unland, "*Issues on Representing Crosscutting Features*", position paper for the Third international workshop on Aspect oriented modeling 2003. http://lglwww.epfl.ch/workshops/aosd2003/papers/Stein-AOMissues.pdf
2. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. Griswold, "*An Overview of AspectJ*", ECOOP'01, Budapest, 2001.
3. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Longtier and John Irwin. "*Aspect oriented programming*". ECOOP'97 LNCS 1241, pp 220-242. Springer-Verlag. June 1997.
4. IBM research. *Hyper/J: Multi-dimensional separation of concerns for Java*. http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm

5. Omar Aldawud, Tzilla Elrad and Atef Bader, "*UML profile for aspect-oriented software development*", position paper for the Third international workshop on Aspect oriented modeling 2003. http://lglwww.epfl.ch/workshops/aosd2003/papers/AldawudAOSD_UML_Profile.pdf

6. Phillip Schmidt, Sergio Alvarado, Jaime Milstein, Gregory Mulert, Robert Duvall and Jesus Rivera, "*A systems engineering perspective of aspect-oriented software architectural analysis using UML*", position paper for the Third international workshop on Aspect oriented modeling 2003.

   http://lglwww.epfl.ch/workshops/aosd2003/papers/Schmidt-SEperspectiveOfAOSAUsingUML.pdf

7. Mark Basch and Arturo Sanchez, "*Incorporating aspects into the UML*", position paper for the Third international workshop on Aspect oriented modeling 2003. http://lglwww.epfl.ch/workshops/aosd2003/papers/Basch-IcorporatingAspectsIntotheUML.pdf

8. *MasterCraft – Component-based Development Environment*. Technical Documents. Tata Research Development and Design Centre. http://www.tcs.com/0_products/mastercraft/index.htm

9. Model driven architecture (MDA). http://www.omg.org/mda/

10. QVT Partners Initial submission to the MOF 2.0 Q/V/T RFP http://www.omg.org/cgi-bin/doc?ad/03-03-27

11. Rich Hilliard, "*Views and viewpoints in software systems architecture*", position paper for the First working IFIP conference on software architecture (WICSA 1), San Antonio, Feb 1999.

12. Robert France, Geri Georg and Indrakshi Ray. "*Supporting multi-dimensional separation of design concerns",* position paper for the Third international workshop on Aspect oriented modeling 2003. http://lglwww.epfl.ch/workshops/aosd2003/papers/Robert-.pdf

13. Siobhán Clarke and Robert J. Walker. "*Composition Patterns: An Approach to Designing Reusable Aspects*" In proceedings of the 23rd International Conference on Software Engineering (ICSE), Toronto, Canada, May 2001.

14. Vinay Kulkarni, R. Venkatesh and Sreedhar Reddy. "*Generating enterprise applications from model"s*. OOIS'02, LNCS 2426, pp 270-279. 2002.

15. Vinay Kulkarni and Sreedhar Reddy. "*Integrating aspects with model-driven software development"*. In proceedings of International conference on Software Engineering Research and Practice, Las Vegas, USA, June 2003.