

UML Aspect Specification Using Role Models

Geri Georg

Agilent Laboratories, Agilent Technologies, Fort Collins, USA
geri_georg@agilent.com

Robert France

Department of Computer Science, Colorado State University
Fort Collins, USA
france@cs.colostate.edu

Abstract. We demonstrate a flexible technique for aspect specification using the UML. The technique uses Role Models to specify design aspects. Roles allow greater flexibility in an aspect over other template-based techniques (e.g. profile extensions). While Role Models do allow us to create templates, they also allow us to create flexible specifications that can be applied by identifying existing model elements that can play aspect roles either as is, or with augmentation based on the aspect specification. This additional capability means that our aspect designs can be applied to specific system designs with fewer constraints on the designer and the initial system models.

We demonstrate this flexibility by applying a design aspect developed for one problem domain to a problem in a different domain. No changes are needed in the aspect models, although not all portions of the aspect specification are used in the second problem. In addition, there is no need to constrain the problem in the new application of the aspect; the specification technique is flexible enough that we can apply the aspect without change. We are also able to use the same set of weaving rules to compose the aspect with models of the new problem.

1 Introduction

Many authors are extending the ideas of aspect-oriented programming (AOP; e.g. [2, 3, 11, 11, 12, 15, 19]) to the architecture and design of complex systems, creating new applications of separation of concerns and aspect-oriented design (AOD; e.g. [1, 9, 13, 16-18]). Some AOD researchers use the Unified Modeling Language (UML; [14]) to specify aspects as well as the models resulting from weaving aspects and other system models (e.g. [4, 5, 8, 10, 20]). Aspect specifications can be viewed as template models, and they are generally woven by using regular expression to match existing model elements and aspect elements. Many aspect compositions essentially result in wrapping additional functionality around an existing model. The proper model factoring must already exist to apply the aspect, so it is conceivable that effort must be applied to re-factor existing models to correctly compose them with aspect models.

Our approach uses Role Models to specify design-level aspects using the UML. Roles are property-oriented specifications, and for a model element to play a particular role it must exhibit the specified properties (e.g. attributes, operation signatures, operation behaviors, and association multiplicities). (See [6, 7] for a complete description of Role Models.) We use UML static diagrams as well as behavioral diagrams to describe aspects. We use the Object Constraint Language (OCL; [14, 21]) to express constraints. We develop weaving rules that allow us to identify model elements in existing system models that can play the required roles in an aspect. The weaving rules also allow us to use aspect roles as either templates (in which case they are added to existing system models), or as extensions that need to be added to existing model elements so that these elements can play the aspect roles. As we discuss in a previous paper [8], our role specifications are based on a more precise and rigorous approach than other AOD methods (e.g. [4, 20]).

The ability to act as a template, or to extend existing model elements, means that our aspects are flexible and therefore fewer constraints need to be placed on their use than if they were strictly templates. We demonstrate this flexibility by applying an example of our aspects (a replicated repositories fault tolerance mechanism developed for a simple order entry system) to a different problem domain (a road traffic pricing system). Section 2 of this paper introduces the fault tolerant aspect. Section 3 demonstrates applying the aspect to a portion of the road traffic pricing system. We conclude in Section 4 with plans for continued work in this area.

2 Fault Tolerance Aspect – Replicated Repositories

We use Static Role Models (SRMs) and Interaction Role Models (IRMs) to specify replicated repositories. SRMs characterize the static structure of our aspect. IRMs characterize interaction diagrams such as collaboration diagrams. This paper only discusses the SRM model of replicated repositories. (We develop and use replicated repository IRMs in [8].) Figure 1 shows the SRM specification of the replicated repository aspect we developed for an Order Entry System (see [8]). In the next section we will see that the form of the aspect facilitates reuse in another application domains.

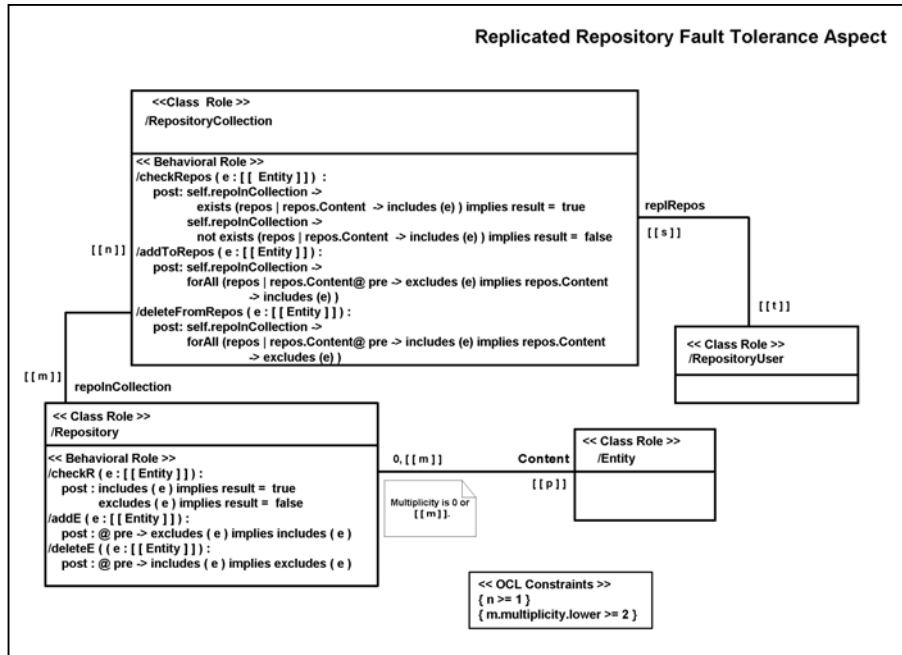


Fig. 1. Static role model specification of a replicated repository aspect.

An SRM consists of roles, and each role has a base type. The base must be either a classifier type (e.g. Class or Interface) or a relationship type (e.g. Association or Generalization). The base indicates the type of UML construct that can play the role. The SRM in Figure 1 consists of two types of roles: class roles (roles with base *Class*) and association roles (roles with base *Association*). The class roles are connected via association roles. Since the role RepositoryCollection shown in Figure 1 has a base Class, only UML class constructs can play this role. The behavioral roles specified in RepositoryCollection determine the additional behavioral properties that must be present in any UML class that plays this role.

A role can consist of two types of properties, metamodel-level constraints and feature roles. Feature roles are further defined as either structural roles (e.g. properties that can be played by class attributes) or behavioral role (e.g. properties that can be played by methods). Feature roles are only associated with classifier roles in our aspect work. While RepositoryCollection does not have behavioral roles, neither it nor any of the other classifier roles in Figure 1 have structural roles.

The association roles shown in Figure 1 are templates; conforming associations can be created by substituting a set of ranges for m , n , p , s , and t that satisfy the OCL constraints shown in the *OCL Constraints* box.

As in our previous paper (see [8]), a model element conforms to a role if it satisfies the metamodel-level constraint and if the constraints associated with its features imply the constraints that would be obtained by instantiating the role as a template. We use a stereotype notation (e.g. <<RepositoryCollection>>) in a woven model to indicate

that the model element realizes that role. Stereotypes used in this manner can result in UML model elements with more than one stereotype.

Figure 1 shows four class roles in the replicated repository fault tolerant aspect. *RepositoryUser* represents the role played by the user of a repository. This user can interact with a replicated repository (an object of a class that plays the *RepositoryCollection* role) in order to access individual repositories (an object of a class that plays the *Repository* role). A replicated repository consists of two or more individual repositories. These repositories contain some replicated content, or entities. Since repositories are replicated, entities are related to more than one repository. Methods that access the repositories in the non-fault tolerant version of the design must be changed (during the weaving process) to access the repository collection in the fault tolerant version. Method post conditions are shown in OCL.

There are three access methods, one to check for the existence of an entity in the repositories, one to add an entity to the repositories, and one to delete an entity from the repositories. The first method returns either true or false, depending on whether there is a repository that contains the entity. The second method adds the entity to each repository in the collection, and the third method removes the entity from all repositories in the collection.

3 Adding the Fault Tolerant Aspect to a Road Traffic Pricing System

Figure 2 shows changes that are made when weaving Replicated Repositories into the core functionality of a portion of a road traffic pricing system. Classes that are added or changed are shown in gray.

Weaving occurs as follows: existing model elements that will play aspect roles are identified and augmented if necessary (e.g. *checkR* and *deleteE* methods must be added to *VehicleOwnerInfo* in order for it to play the role of *Repository*), new model elements are added for any aspect roles that do not exist in the original model (e.g. *VOReplRepo*, *DSReplRepo*, and *Summaries*), relationships that do not exist in the original model but do exist in the aspect must be added (e.g. the relationship between *VOReplRepo* and *VehicleOwnerInfo*), and relationships that exist in the original models but do not exist in the aspect must be deleted, and finally, multiplicities on relations must be reconciled between aspect templates and the original models (e.g. the 1 and 2..* multiplicities on the relation between *VOReplRepo* and *VehicleOwnerInfo*).

4 Conclusions and Future Work

We demonstrate the flexibility of our specification of aspects using Role Models by applying an aspect developed for a particular application to another application in a different domain. We continue to develop additional aspect models for fault tolerance and other cross-cutting concerns such as security. In addition, we are developing a

prototype tool to support the weaving process, including the ability for the user to specify multiple weaving strategies.

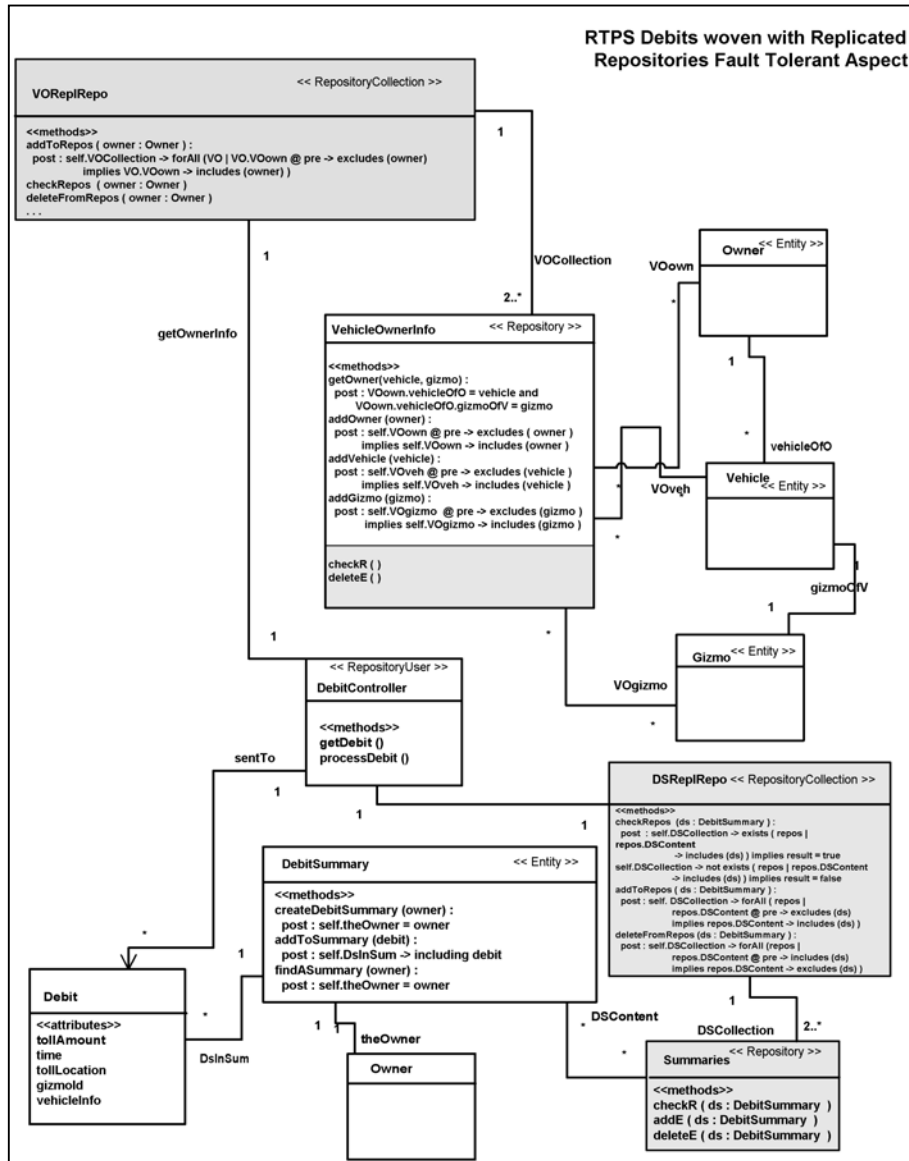


Fig. 2. Road Traffic Pricing System static diagram with replicated repositories.

References

1. **Andrade, L. F. and J. L. Fiadeiro.** 2001. Coordination technologies for managing information system evolution. Proceedings CAISE'01. LNCS, Springer-Verlag.
2. **Bergenti, F. and A. Poggi.** Promoting reuse in aspect-oriented languages by means of aspect views.
3. **Bergmans, L. and M. Aksit.** 2001. Composing crosscutting concerns using composition filters. Communications of the ACM **44**(10, October):51-57.
4. **Clarke, S. and R. J. Walker.** 2001. Composition patterns: an approach to designing reusable aspects.
5. **Fiadeiro, J. L.** Co-ordination based development and evolution.
6. **France, R. B., D. K. Kim, and E. Song.** 2002. Patterns as precise characterizations of designs. Technical Report 02-101, Computer Science Department, Colorado State University.
7. **France, R., D. K. Kim, E. Song, and S. Ghosh.** 2001. Using roles to characterize model families. Proceedings of the Tenth OOPSLA Workshop on Behavioral Semantics: Back to the Basics.
8. **France, R. and G. Georg.** 2002. Modeling fault tolerant concerns using aspects. submitted to ISSRE 2002.
9. **Gray, J., T. Bapty, S. Neema, and J. Tuck.** 2001. handling crosscutting constraints in domain-specific modeling. Communications of the ACM **44**(10, October):87-93.
10. **Jurjens, J.** 2001. Towards development of secure systems using UMLsec. 4th International Conference on Fundamental Approaches to Software Engineering (FASE 2001):187-200.
11. **Kiczales, G., E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold.** 2001. Getting started with AspectJ. Communications of the ACM **44**(10, October):59-65.
12. **Lieberherr, K., D. Orleans, and J. Ovlinger.** 2001. Aspect-oriented programming with adaptive methods. Communications of the ACM **44**(10, October):39-41.
13. **Netinant, P., T. Elrad, and M. E. Fayad.** 2001. A layered approach to building open aspect-oriented systems. Communications of the ACM **44**(10, October):83-85.
14. **Object Management Group.** 2001. Unified Modeling Language V. 1.4. <http://www.omg.org>, September.
15. **Ossher, H. and p. Tarr.** 2001. Using multidimensional separation of concerns to (re)shape evolving software. Communications of the ACM **44**(10, October):43-50.
16. **Pace, J. A. D. and M. R. Campo.** 2001. Analyzing the role of aspects in software design. Communications of the ACM **44**(10, October).
17. **Silva, A. R.** Separation and composition of overlapping and interacting concerns.
18. **Silva, A. R.** 1999. Separation and composition of overlapping and interacting concerns. In OOPSLA '99, Multi-Dimensional Separation of Concerns in Object-Oriented Systems. <http://www.esw.inesc.pt/dasco/>.
19. **Sullivan, G. T.** 2001. Aspect-oriented programming using reflection and metaobject protocols. Communications of the ACM **44**(10, October):95-97.
20. **Suzuki, J. and Y. Yamamoto.** 1999. Extending UML with aspects: Aspect support in the design phase. Proceedings of the third ECOOP Aspect-Oriented Programming Workshop.
21. **Warmer, J. and A. Kleppe.** 1999. The Object Constraint Language, Addison Wesley Longman, Inc.