Composing Aspect Models

Geri Georg, Robert France, and Indrakshi Ray
Department of Computer Science
Colorado State University, Fort Collins, CO 80523

Abstract.

Aspect-oriented modeling (AOM) techniques allow system developers to address pervasive objectives such as security and fault-tolerance separately from core functional requirements during system design. An aspect-oriented design model consists of a set of aspects and a primary model. An aspect describes how a single objective is addressed in a design, and a primary model describes how core functionality requirements are addressed. In order to analyze the interactions between aspects and primary models they must be composed. System developers may need to create and analyze alternative realizations in order to produce a design that balances competing objectives (concerns). Treating realizations of design objectives as aspects allows developers to more easily swap in and out alternative realizations in a design.

The iterative nature of design dictates that composition and analysis be carried out in a flexible and intuitive manner. Composing aspects and primary models can produce designs with conflicting structures or behaviors. We have developed a two-level structure of composition constraints to address this issue; a high level that identifies the aspects and determines the order in which they will be composed (composition strategy), and a lower level that constrains how a single aspect is composed with a primary model (composition directives). In this paper we describe a model composition approach that utilizes composition constraints. We illustrate the approach using small examples of security and fault-tolerance aspects.

1. Introduction

Problems and their solutions are decomposed to manage complexity during system development. Decisions made in the early stages of design identify objectives that determine the primary structure of the design. Given a primary structure, it may not be possible to localize information about how other equally important concerns (e.g., security and fault tolerance objectives) are addressed in design units. Using current design modeling techniques, these objectives are addressed by functionality that is distributed across the design. The distributed nature of this functionality makes it difficult to understand, analyze, and change the functionality. In the cases where objectives compete, the distributed functionality can make it difficult to consider and evaluate alternative realizations of the objectives.

The Aspect-Oriented Modeling (AOM) technique we are developing allows developers to localize cross-cutting realizations of concern objectives in aspects. An AOM design consists of one or more aspects and a primary model. Aspect models describe behavior that cross-cuts the primary model. In order to obtain an integrated design view aspect and primary models must be composed. The composed model is needed to support design analysis that can uncover inconsistencies arising from conflicting structures or behaviors defined in aspect and primary models. A conflict can occur when a property in an aspect (e.g. a multiplicity, or the presence or absence of an attribute or relation) contradicts a property in another aspect or in the primary model. Conflicts can also occur when behavior defined by an aspect cannot be performed as specified because some of its sub-behaviors have been modified (or deleted) after merging with behaviors defined in other aspects or the primary model. In these cases, conflict resolution requires intervention by the system developer. We have found that some conflicts can be resolved by constraining how aspects are composed with primary models. As part of our AOM work, we are developing composition tools and techniques that allow developers to constrain how composition is performed.

The usability and utility of the AOM approach is greatly enhanced if significant portions of the composition activity are automated. At one extreme are composition tools that take in aspect and primary models and produce composed models without further input from developers. This fixed composition approach provides very little flexibility in how aspect models are composed with primary models. At the other extreme are composition tools that require developers to specify how the aspect models are to be composed with primary models. This approach is very flexible, but requires more effort from developers. More practical solutions are likely to lie between these two approaches. We are developing a composition tool that implements a basic composition procedure but that also allow developers to vary some aspects of the composition using composition constraints. In this paper we discuss how composition constraints can be used to vary composition such that conflicts are minimized.

The rest of the paper is structured as follows. Section 2 provides a brief overview of the UML modeling techniques we use to specify both the structure and behavior of aspects. Section 3 discusses the types of conflicts we have encountered during model composition and the composition constraints (strategies and directives) we have developed to cope with these conflicts. Section 4 discusses related work, and Section 5 presents conclusions and future work.

2. Modeling design aspects using the UML

The AOM approach we are developing can be used to localize a cross-cutting realization of an objective when the distributed parts of the realization have common structural and behavioral characteristics. This allows the developer to treat the realization as a pattern that is instantiated in different parts of the design. An aspect is thus a pattern of structure and behavior [17]. An aspect model consists of UML template diagrams that describe the pattern from structural and behavioral perspectives. A primary model consists of a set of UML diagrams describing realizations of core functional requirements.

Typically, an aspect model consists of a UML class diagram template and one or more interaction diagram templates. The class diagram template is used to generate class diagrams that describe the structures that are distributed across the design. Interaction diagram templates are used to generate interaction diagrams that describe how elements in the distributed structures interact to realize the desired behavior. A template element specifies properties that are to be incorporated into user designated points of a primary model. Each template element is a potential integration point with a corresponding element in the primary model. The specific integration points with a primary model are not specified as part of the generic aspect model. These points are specified separately as mapping rules. This allows considerable flexibility in using and reusing aspects

Composition of an aspect model and a primary model is a two-stage process. In the first stage the template diagrams in an aspect model are instantiated using mapping rules that determine the model elements passed in as parameters to the templates. If a parameter value is a primary model element then it represents a point of integration with the primary model. If a parameter value is not a primary model element then it represents a design element that is to be added to the design during composition. The result of the first stage is a set of UML diagrams collectively called the context-specific aspect model. The context-specific aspect model presents a view of the design that describes how the original objective is addressed. In the second stage the context-specific aspect model is merged with the primary model. Elements with matching names and properties are merged to produce a composed model. (Note that although composition proceeds using elements with matching names and thus provides syntactic composition, it also uses element properties including those expressed in OCL, and thus also provides some semantic composition.) The developer can use composition strategies and directives to drive how the models are merged as discussed in the next section.

3. Model composition

From our work on modeling pervasive security and dependability objectives as aspects it is apparent that the combination of aspects chosen to address objectives, the order in which the aspects are composed, and the manner in which individual model elements are composed all determine whether the composed model will meet its objectives. We have identified two types of composition constraints that can be defined by developers to influence the composition process. *Composition strategies* use high-level heuristics to identify a set of aspects and to determine how the aspects are composed with a primary model in order produce a design that satisfies specified dependability or security goals. The heuristics provide answers, based on experience, to questions such as “Is it enough to add encryption to a system to ensure privacy?” (Possible answer: Probably not; authentication and perhaps access control are also needed.); “Does data replication always ensure availability?” (Possible answer: Probably not; it depends upon the environment in which the system is deployed.). We presented heuristics in a previous paper (see [4]) that can be used to determine the aspects that should be composed with a primary design to fulfill certain security goals.

Composition strategies can be suggested by problem domain knowledge, the physical configuration of a system, or prior experience and the result of trade-off analyses. For example, in a system where certain data is considered confidential, particular aspects need to be composed with the functionality surrounding that data in order to protect it (problem domain knowledge). If the data will be accessed over an un-trusted communication link, further authentication and encryption aspects may be needed (physical environment). Finally, prior experience may show that complete protection in such an environment is not possible, so auditing and recovery aspects may need to be composed to part of the system. Prior experience also dictates that if auditing is added, special care must be taken to ensure that all the functionality that should be audited is audited. (In this case a simple ordering of aspect composition, with auditing being last may be sufficient to realize the desired behavior.)

In addition to composition strategies, specific directives can often be used to prevent the conflicts that can arise from merging a context-specific aspect model and a primary model. We call these *composition directives*. For example, a typical default composition action is to conjugate class attributes and operations. In some situations this may result in the desired behavior of an operation, but in others it can cause conflict. As an example of how composition directives can be used to resolve such conflicts consider the case in which the desired result of composing an authentication aspect with a primary model is an operation in the primary model, named *OP*, that is *wrapped* with the authorization behavior defined in the aspect. The authorization behavior in the context-specific aspect is defined in terms of two operations: an operation named *OP* that carries out the authorization check and then calls another operation called *doOP* to perform some task for authorized clients. To merge these operations to get the desired result and to avoid merging the conflicting *OP* definitions, the *OP* operation in the primary model needs to be renamed to *doOP*. The result is a composed model in which *OP* performs the authorization check, and then calls *doOP* if the check reveals that the client is authorized to invoke the behavior. A *composition directive* is used to rename the operation in the primary model before composing the operations. The renaming removes the conflict and allows the primary operation to be properly composed with the operation in the authorization aspect.

Another typical default composition action is to use the stronger multiplicity at an association end as the multiplicity at the corresponding association end in the composed model. An example of a conflict that can arise from this default composition is when a replicated repository aspect is composed with a primary model that contains an association from a class to a single copy of a repository. In the aspect view the class has an association to two or more repositories. Merging these two view results in a conflict: the multiplicity at the repository end in the primary model is *1*, and in the aspect model it is *2..**. In this case a composition directive can be used to override the primary model multiplicity with the aspect model multiplicity.

In our composition technique then, a directive can (1) define precedence or override relationships between matching aspect and primary model elements with conflicting properties or definitions, (2) rename elements to resolve conflicts, and (3) specify elements that are to be added or are deleted in a composition.

In summary, composition strategies help system developers decide what aspects need to be composed with a primary model to realize system design goals, and composition directives allow developers to vary how aspect and primary models are composed. A consequence of allowing the use of composition directives is that aspect models do not need to capture all possible variations (thus simplifying their definitions). For example, composition directives can be used to modify an aspect model to obtain a variant that is more suitable for the context in which it will be used. Composition directives thus allow flexibility in the way models are composed. At the same time, defining defaults for each of the composition actions allows the process to be simplified.

4. Related Research

Aspect-Oriented Development (AOD) supports the separation of concerns principle that has proven to be effective at tackling complexity [6]. AOD methods allow developers to represent pervasive design and implementation concerns as *aspects*. In an AOD approach, a design consists of (1) a primary design or implementation artifact (e.g., a UML model or code) in which the pervasive concerns are not included, (2) a set of aspects, each representing a pervasive design concern that impacts the elements of the primary design artifact, and (3) a weaving mechanism that composes aspects with the primary artifact to obtain a view of the design that details how the structures and behaviors modeled in the primary artifact are impacted by the aspects. Examples of AOD approaches are *aspect-oriented programming* (e.g., see [1, 8, 9, 10, 11, 14, 15]) in which the primary design artifacts are code, and aspects are concerns that cross-cut code modules, and *subject-oriented design* (e.g. see [2, 3, 7, 16]) in which aspects are design realizations of requirements, and a design is created by composing aspects. In our AOM approach, aspects are design realizations of security and dependability objectives. We do not treat the design realizations of all requirements as aspects.

The subject-oriented design approach proposed by Clarke et al. is a UML-based approach that is closest to our AOM method [2]. In the subject-oriented modeling approach a design is created for each system requirement. The design for a system requirement is referred to as a *subject*. A comprehensive design is obtained by composing subjects. In the subject-oriented approach aspects are subjects expressed as UML model views, and composition involves merging the views provided by the subjects. Merging is restricted to adding and overriding named elements in a model. Merging of constraints is not supported, nor is there support for deleting elements from models (except the implicit deletion that occurs when an element is overridden). Conflict resolution mechanisms are limited to defining precedence and override relationships between conflicting elements. In prior work [4, 5] we have shown how security and dependability characteristics can be modeled as design aspects, expressed as structural and behavioral patterns specifications, and woven into designs expressed in the UML. We have also demonstrated some conflicts that can occur during composition, and directives that can be used to resolve them.

As part of the Early Aspects initiative, Moreira, Araujo, and Rashid have targeted multi-dimensional separation beginning early in the software cycle [12, 13]. Their work supports modularization of broadly scoped properties at the requirements level to establish early trade-offs, provide decision support and promote traceability to artifacts at later development stages. Our AOM method complements this work by supporting aspect modeling, composition, and analysis of successively more detailed levels of abstraction needed during system design.

5. Conclusions and Future Work

Aspect-oriented modeling provides a straightforward way to model, compose, and analyze multiple competing critical systems objectives such as security and dependability. AOM allows system developers to design these concerns separately, and then compose them with primary models to create integrated models that can be analyzed. Developers can then use analysis results to drive further changes to the design of competing concerns. Composition and analysis must be powerful, yet flexible and easily understood to enable this iterative process.

We have developed a two-level process for constraining composition in our AOM method. The first level is based on heuristics that help an architect or system developer decide which particular aspect models should be composed with primary models in order to meet system design goals. The second level allows the developer to specify particular directives to drive the composition of all or portions of the aspect model with the primary model. The combined two-level approach has been successful in preventing common model composition conflicts.

We continue to work in this area to develop a terminology that will codify composition strategies, and allow their inclusion into an automated prototype tool. We are also developing notation for stating composition directives.

6. References

[1] Bergmans, L. and M. Aksit, M., "Composing multiple concerns using composition filters", *Communications of the ACM*, vol 44, no 10, Oct 2001

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

[3] Fiadeiro, J. L. and Lopes, A., "Algebraic semantics of co-ordination or what is it in a signature?", *Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology (AMAST'98)*, Amazonia, Brasil, Lecture Notes in Computer Science, vol 1548, pp 293-307, A. Haeberer , ed, Springer-Verlag, Jan 1999

[4] Georg, Geri, France, Robert, and Ray, Indrakshi, "Designing High Integrity Systems using Aspects", *Proceedings of the Fifth IFIP TC-11 WG 11.5 Working Conference*

on Integrity and Internal Control in Information Systems (IICIS 2002), Bonn, Germany, Nov 2002

[5] Georg, G., Ray, I., and France, R., "Using Aspects to Design a Secure System", *Proceedings of the Interational Conference on Engineering Complex Computing Systems (ICECCS 2002)*, ACM Press, Greenbelt, MD, Dec 2002

[6] Ghezzi, C., Jazayeri, M., and Mandrioli, D., Fundamentals of Software Engineering, Prentice Hall, 1991

[7] Gray, J., Bapty, T., Neema, S., and Tuck, J., "Handling crosscutting constraints in domain-specific modeling", *Communications of the ACM*, vol 44, no 10, pp 87-93, Oct 2002

[8] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G., "An Overview of AspectJ", *Proceedings of the European Conference on Object-Oriented Programming (ECOOP '01)*, pp 327-353, Budapest, Hungary, June, 2001

[9] Kieberherr, K., Orleans, D., and Ovlinger J., "Aspect-oriented programming with adaptive methods", Communications of the ACM, vol 44, num 10, pp 39-41, Oct 2001

[10] Ossher, H. and Tarr, P., "Using multidimensional separation of concerns to (re)shape evolving software", *Communications of the ACM*, vol 44, num 10, p 43-50, Oct, 2001

[11] Pace, J. A. D. and Campo, M. R., "Analyzing the Role of Aspects in Software Design", *Communications of the ACM*, vol 44, no 10, pp 66-73, Oct 2001

[12] Rashid, A., Moreira, A., and Araujo, J., "Modularization and Composition of Aspectual Requirements", *2nd International Conference on Aspect-Oriented Software Development*, ACM, pp 11-20, Boston, Mar 2003

[13] Rashid, A., Sawyer, P., Moreira, A., and Araujo, J., Early Aspects: A Model for Aspect-Oriented Requirements Engineering, *IEEE Joint International Conference on Requirements Engineering*, IEEE Computer Society Press, pp 199-202, Essen, Germany, Sept 2002

[14] Silva, A. R., "Separation and composition of overlapping and interacting concerns", *OOPSLA '99 First Workshop on Multi-Dimensional separation of Concerns in Object-Oriented Systems*, Denver, Colorado, Nov 1999

[15] Sullivan, G. T., "Aspect-oriented programming using reflection and metaobject protocols", *Communications of the ACM*, vol 44, num 10, pp 95-97, Oct 2001

[16] Suzuki, J. and Yamamoto, Y., "Extending UML with Aspects: Aspect Support in the Design Phase", *Proceedings of the 3rd ECOOP Aspect-Oriented Programming Workshop*, Lisbon, Portugal, June 1999

[17] The Object Management Group, "The Unified Modeling Language", OMG, formal/2001-09-67, version 1.4, 2001