

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 concerns such as security, fault-tolerance, safety, and availability separately from core functional requirements during system design. Such separation implies that developers must also be able to analyze the integration of these additional concern designs with core functionality designs in order to compare realizations that may be based on conflicting design goals. System developers may create and analyze alternative design realizations several times before a balanced system design is achieved. Separating concerns as aspects can aid in this design evolution since a developer can focus on the realization of a single concern without having to worry about its interactions with other concerns or primary functionality.

However, the iterative nature of the process dictates that the composition and analysis of the resulting integrated design must be flexible as well as intuitive. We have found that composing aspect design models and primary models often leads to conflicting structure or behavior in the integrated model in all but the simplest cases. This is particularly true when detailed models are being composed. We have developed a two-level structure of composition directives to address this issue; a high level that dictates the conditions under which specific aspects should be integrated to realize particular design concerns, and a lower level that precisely defines which portions of particular models should be integrated in specific ways. At the same time we note that composition directives must not be so numerous or complex that they cannot be easily understood and specified by system developers desiring to perform model composition.

In this paper we describe model composition techniques that allow flexibility in an automated approach. This work is part of our aspect-oriented modeling (AOM) method that supports localizing cross-cutting realizations of design concerns in aspect models. We illustrate our composition technique using small examples of security and fault-tolerance concern design realizations.

1. Introduction

Decomposition of problems and their solutions is used to manage complexity during software system development. Decomposition can be used to partition multiple competing system design concerns. Decisions made early in the design process identify some of these concerns as most crucial to the system, and realizing these concerns determines the modular structure, or primary decomposition of the design. Other concerns that impact how the primary functionality is delivered to users (e.g. privacy and availability) may be equally important to meeting system goals. The way these additional concerns are realized can impact each other, and sometimes the primary system functionality since they can interfere with each other and/or the primary functionality. System architects must therefore trade off these additional concerns against each other to create an optimal system design that best meets the overall desired behaviors of the system.

Aspect-Oriented Modeling (AOM) techniques can support this effort since they allow system developers to separate designs as aspects, without having to worry about the interactions of the concern realization with those of other concerns or the primary decomposition design. However, since interactions often do occur, developer must also have techniques at their disposal to integrate all designs and analyze the results for undesirable interactions. These interactions, or conflicts, can be resolved by an iterative process of changing the designs (either the concern realizations or the primary decomposition), re-composing the designs, and analyzing the result. A key element of this iterative process is a set of mechanisms that easily and powerfully integrate aspect models with primary decomposition models. We have found that in all but the simplest cases, aspect model composition can lead to conflicts in the resulting integrated model. Conflicts arise from two sources. A conflict can occur when a property in one aspect model (e.g. a multiplicity, or the presence or absence of an attribute or relation) contradicts a property in another aspect model. Conflicts can also occur when behavior defined by one aspect model cannot be

performed as specified because some of its sub-behaviors have been modified (or deleted) after merging with behaviors defined in other aspect models or the primary model. In either case, the system developer must resolve the conflict manually. We are developing tools and methodologies as part of our AOM program at Colorado State University, and are therefore interested in automating the composition process as much as possible. We have found that the way aspect models are integrated with primary models can often resolve some of these conflicts.

We are developing a composition technique that can be supported by tools to automate significant parts of the composition activity. Such automation is necessary if AOM approaches are to be useful and usable in industrial software development environments. 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, and often produces conflicts of the types discussed above in the composed model. At the other extreme, composition tools can require developers to provide composition procedures that detail exactly 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: composition tools are needed that codify composition procedures but that also allow developers to vary some aspects of model integration. This is the approach taken in our work.

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 our aspect models. Section 3 discusses the types of conflicts we have encountered during model composition and the weaving 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

We define design aspect models in terms of structures of template UML diagrams (see [34]). These template diagrams were developed from *Role Models* we initially used to specify behavioral and structural properties captured by design patterns [8, 9]. The template form of UML diagrams facilitates tool-supported composition of design aspect models and primary models. Template aspect models are generic in that they can be composed with any number of primary decomposition models by instantiating the templates with specific parameter names taken from the parameter model. Composition must therefore occur in two phases.

In the first phase, a generic template model must be instantiated to create a context-specific UML aspect model with (some) primary decomposition model elements substituted for template parameters. Template parameters (identified with a “[” in front of the parameter name) may be instantiated using comparable model element names from the primary model being integrated, or if no such element exists, new element names are used to create the context-specific aspect model. The second phase of composition consists of merging the context-specific model with the primary decomposition model according to composition directives.

The templates in our aspect models specify the properties that are to be incorporated into user designated points of a primary model. Each model element can be considered a potential integration point with the primary model. Thus, each generic design aspect model specifies its own integration model, which allows considerable flexibility and reuse in aspect model definitions. The specific integration points with a primary model are not specified as part of the generic aspect model. Instead, the primary model elements used to instantiate a context-specific aspect model during the first phase of composition become the integration points between the aspect model and the primary decomposition model.

We use both structural and behavioral diagrams to specify aspect models. We have developed algorithms to compose structural specifications written using the UML as class diagrams and behavioral specifications as UML 1.4 collaboration diagrams. We are in the process of developing composition algorithms for UML 2.0 sequence diagrams as well. Due to space constraints, the rest of this paper only discusses composition of aspect structural specifications with primary system structural designs.

3. Model composition

We have found that most of the work relating to composition deals with the mechanics of composing specific portions of aspect models. However, in our work with security and dependability concerns it is apparent that there is a more abstract level of composition that must be addressed. This level deals with what types of aspects need to be composed with a primary model to achieve specific dependability or security goals. For example, is it enough to add encryption to a system to ensure privacy? (Probably not; authentication and perhaps access control are also needed.) Does data replication always ensure availability? (Again, probably not but it depends upon the environment in which the system is deployed.) We presented heuristics in a previous paper (see [13]) that can be used to determine when particular aspects should be composed with a primary design to fulfill certain security goals. We call these high-level heuristics *composition strategies*.

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.) A significant challenge is to develop a language for expressing composition strategies and techniques for obtaining composition directives from strategies. We are currently addressing these problems in our AOM research.

In addition to composition strategies, specific directives can often be used to prevent the conflicts that can arise from composition of individual model elements in the context-specific aspect model and the primary decomposition model. We call these *composition directives*.

Due to space constraints, we only discuss the composition of structural models in this paper. (We have developed a similar composition rules for behavioral specifications, in particular for UML 1.4 collaboration diagrams. We are also working on composition rules for UML 2.0 sequence diagrams.) In general, composition of model elements proceeds as follows:

1. All primary model elements that have the same name and type as a prohibited element in the context-specific aspect model are not included in the composed model.
2. If the matching elements are classifiers, then the classifier elements are merged. If a classifier element in the context-specific aspect model does not have the same name as any element of the same type in the primary model classifier then the context-specific aspect model element is added to the composed model classifier. Similarly, if a classifier name in the primary model does not have the same name as any element of the same type in the context-specific model classifier then the primary model element is added to the composed model classifier.
3. If the matching elements are associations, then the stronger multiplicity at an association end is used as the multiplicity at the corresponding association end in the composed model.
4. If the matching elements are operations or attributes with constraints, the elements are conjugated in the composed model. By default then, pre-conditions and post-conditions of operations are *anded* in the woven model. Similarly, attribute constraints are *anded* in the woven model.

Naïve application of these rules often results in conflicts. For example, the desired result of composing an authentication aspect with a primary model may be that an operation in the primary model is basically *wrapped* with an authentication operation. The name of the wrapping operation thus needs to be the same as the name of the original operation, and the original operation needs to be renamed. If composition rule 4 described above is applied, the result is an operation that merely conjugates both operations. A *composition directive* needs to be 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 authentication aspect.

Another example of a conflict that can arise from the composition rules is when a replicated repository aspect is composed with a primary model that contains a relation from a class to a single copy of a repository. The multiplicity in the primary model is 1 , and in the aspect model it is $2..*$. If composition rule 3 is applied as described above, the more restrictive multiplicity of 1 will be placed in the composed model. The conflict arises from the fact that the intent of the replication aspect is to have more than one repository, so in this case, the aspect relation multiplicity needs to override the primary model multiplicity.

In our composition technique then, a directive can (1) define conjunction, disjunction, precedence, or override relationships between matching aspect and primary model elements with conflicting properties or definitions, or (2) determine the elements that are renamed to resolve conflicts, are added, or are deleted. (Adding new elements may be necessary to correctly integrate the behavior defined by the aspect model into the particular primary decomposition model.)

In summary, composition strategies help system developers decide what aspects need to be composed with primary functionality 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 rules allows the process to be simplified.

4. Related Research

Model-Based Development (MBD), sometimes referred to as Model-Driven Development (MDD, see [2]) is concerned with using models as the primary artifacts of software development, and is supported by techniques for rigorously analyzing models and for generating production-strength implementations from models. MBD methods raise the level of abstraction at which developers compose and implement systems. Examples of MBD approaches are (1) the Model Driven Architecture (MDA) initiative of the Object Management Group (OMG) that emphasizes the use of the Unified Modeling Language, model transformations, and code generation (see <http://www.omg.org/mda>); and (2) Model-Integrated Computing (MIC) for embedded systems that emphasizes rigorous analyses of models and code generation (see <http://www.isis.vanderbilt.edu/research/research.html>). Our AOM method directly enables these model-based approaches since it is based on standard modeling techniques.

Aspect-Oriented Development (AOD) supports the separation of concerns principle that has proven to be effective at tackling complexity [17]. 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, 19, 20, 21, 22, 25, 26, 31, 32]) in which the primary design artifacts are code, and aspects are concerns that cross-cut code modules, and *subject-oriented design* (e.g. see [3, 4, 5, 18, 33]) in which aspects are design realizations of requirements, and a design is created by composing aspects. Our AOM method is related in that it addresses design realizations of important system concerns. The AOM method differs in that we concentrate on system concerns other than those involved in the primary system modularization. We also concentrate our efforts on using aspect models to support design analysis and trade-off analysis among competing system concerns.

Fiadeiro et al. [5] specify secondary system characteristics related to system coordination using an algebraic approach. Their approach is applicable to detailed design and code, and utilizes a notation that is not widely known by system developers. Gray et al. [18] use aspects to represent secondary system characteristics in domain-specific models. Their research is part of the MIC initiative that targets embedded software systems specifically. Model-Integrated Computing (MIC) extends the scope and usage of models such that they form the backbone of a development process for building embedded software systems. Requirements, architecture, and the environment of a system are captured in the form of formal high-level

models that allow the representation of concerns. Suzuki et al. [33] extend the UML so that it can be used to model code level aspects. Their approach is restricted to secondary system characteristics that can be represented as aspects in an aspect-oriented program. Our approach differs since we do not require aspect-oriented programming techniques.

The subject-oriented design approach proposed by Clarke et al. is a UML-based approach that is closest to the AOM method [3, 4]. 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 [6, 13, 14, 16] we have shown how secondary system 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 [27, 28, 29, 30]. 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 compliments this work by supporting aspect modeling, composition, and analysis of successively more detailed levels of abstraction needed during system design. To our knowledge, our work is unique in this respect.

5. Conclusions and Future Work

Aspect-oriented modeling provides a straightforward way to model, compose, and analyze multiple competing critical systems concerns such as security and dependability. AOM allows system developers to design these concerns separately, and then compose them with primary functionality 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. Easy experimentation with different concern realizations can in turn allow system developers to quickly choose the concern realizations that best meet overall system goals.

We have developed a two-level composition process 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 decomposition 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 algorithms and notations for specific composition directives. This work is also being integrated into our prototype tool.

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] Booch, G., "Growing the UML", *Software and System Modeling Journal*, Vol 1, no 2, Feb 2003
- [3] 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
- [4] Clarke, S. and Murphy, J., "Developing a tool to support the application of aspect-oriented programming principles to the design phase", *Proceedings of the International Conference on Software Engineering (ICSE '98)*, Kyoto, Japan, April, 1998

- [5] 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
- [6] France, R. and Georg, G., Modeling fault tolerant concerns using aspects, TR02-102, Computer Science Department, Colorado State University, 2002
- [7] France, R., Georg, G., and Ray, I., "Supporting Multi-Dimensional Separation of Design Concerns", *2nd International Conference on Aspect-Oriented Software Development: Aspect-Oriented Modeling with UML Workshops*, March, 2003
- [8] France, R. B., Kim, D. K., and Song, E., Patterns as Precise Characterizations of Designs,
- [9] France, R. B., Kim, D. K., Song, E., and Ghosh, S., "Using Roles to Characterize Model Families", in Practical Foundations of Business and System Specifications, Haim Kilov, ed., Kluwer Academic Publishers, 2002
- [10] Georg, G., Run-Time Configuration Management Specification for Distributed Systems, Ph.D. Dissertation, Computer Science Department, Colorado State University, 2001
- [11] Georg, G., Bieman, J., and France, R., "Using Alloy and UML/OCL to Specify Run-Time Configuration Management: A Case Study", *Proceedings of UML Workshop on the Practical UML-Based Rigorous Development Methods*, GI-Edition, Lecture Notes in Informatics, Oct 2001
- [12] Georg, G., France, R., "UML Aspect Specification Using Role Models", *Advances in Object-Oriented Information Systems: OOIS2002*, Sept, 2002
- [13] 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
- [14] Georg, G., and France, R., and Ray, I., "An Aspect-Based Approach to Modeling Security Concerns", *Proceedings of the Workshop on Critical Systems Development with UML*, Dresden, Germany, Oct 2002
- [15] Georg, G., Seidman, S., "The Use of Architecture Description Languages to Describe a Distributed Measurement System", *The Engineering of Computer Based Systems*, Edinburgh, Scotlan, April, 2000
- [16] 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
- [17] Ghezzi, C., Jazayeri, M., and Mandrioli, D., Fundamentals of Software Engineering, Prentice Hall, 1991
- [18] 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
- [19] 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
- [20] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G., "Getting started with AspectJ", *Communications of the ACM*, vol 44, num 10, pp 59-65, Oct, 2001
- [21] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J-M., and Irwin, J., "Aspect-Oriented Programming", *Proceedings of the European Conference on Object-Oriented Programming (ECOOP '97)*, pages 220-242, Lecture Notes in Computer Science, vol 1241, Jyvaskyla, Finland, June, 1997
- [22] 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
- [23] Mekerke, F., Georg, G., France, R., Alexander, R., "Tool Support for Aspect-Oriented Design", *Advances in Object-Oriented Information Systems: OOIS2002 Workshops*,
- [24] Mellor, Stephen and Balcer, Marc, Executable UML: A Foundation for Model Driven Architecture, Addison Wesley Professional, 2002
- [25] 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
- [26] 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
- [27] Rashid, A., "A Hybrid Approach to Separation of Concerns: The Story of SADES", *3rd International Conference on Meta-Level Architectures and Separation of Concerns (Reflection)*, Springer-Verlag Lecture Notes in Computer Science, vol 2192, pp 231-249, Kyoto, Japan, Sep 2001

- [28] Rashid, A. and Chitchyan, R., "Persistence as an Aspect", *2nd International Conference on Aspect-Oriented Software Development*, ACM, pp 120-129, Boston, Mar 2003
- [29] 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
- [30] 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
- [31] 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
- [32] Sullivan, G. T., "Aspect-oriented programming using reflection and metaobject protocols", *Communications of the ACM*, vol 44, num 10, pp 95-97, Oct 2001
- [33] 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
- [34] The Object Management Group, "The Unified Modeling Language", OMG, formal/2001-09-67, version 1.4, 2001
- [35] Wu, T., "The secure remote password protocol", *Proceedings of the 1998 Internet Society Symposium on Network and Distributed Systems Security*, pp 97-111, San Diego, CA, March 1998
- [36] Wu, T., "The SRP authentication and key exchange system", RFC 2945, Network Working Group, 2000