

Modeling Aspect-Oriented Compositions

Thomas Cottenier^{1,2}, Aswin van den Berg¹, Tzilla Elrad²

¹ Software and Systems Engineering Research, Motorola Labs
1300 E Algonquin Road, 60173 Schaumburg, IL, USA

{thomas.cottenier, aswin.vandenberg}@motorola.com

² Concurrent Programming Research Group, Illinois Institute of Technology
3300 S Federal Street, 60616 Chicago, IL, USA

{cotttho, elrad}@iit.edu

Abstract. Crosscutting concerns are pervasive in embedded software, because of the various constraints imposed by the environment and the stringent QoS requirements on the system. This paper presents a framework for modularizing crosscutting concerns in embedded and distributed software, and automating their composition at the modeling level, for simulation and validation purposes. The proposed approach does not extend the semantics of the UML in order to represent aspects. Rather, it dedicates a metamodel to the representation of the composition semantics between aspects and core models. The paper illustrates this approach by presenting a model weaver for SDL statecharts developed at Motorola Labs. Crosscutting behavior is designed with plain SDL statecharts and encapsulated into modules called aspect beans. The weaver looks at the aspect beans and the core SDL statecharts from a perspective that is defined by lightweight extensions to the SDL and UML metamodels. A connector metamodel defines the structure of the aspect-to-core binding definition. Finally, a weaver behavioral metamodel defines composition primitives for specifying weaving strategies.

1 Introduction

Model-Driven Development (MDD) is a software development methodology that emphasizes precise modeling for automated generation of optimized code. MDD techniques also target automated simulation, verification, use-case validation and test case generation. MDD approaches have been widely adopted in the industry, especially for the development of applications that have stringent performance requirements, such as embedded control software for telecommunication devices.

Embedded Software applications are especially difficult to design and build because of the constraints placed on them and because of the problem domain. The telecommunication industry has therefore driven a research effort towards a standard specification language for real-time, stimulus response systems, the Specification and Description Language (SDL) [1]. SDL is standardized as an ITU recommendation, and has continually evolved since its first version in 1980 to now include Object-Oriented features. In SDL, the basis for description of behavior is communicating Extended State Machines that are represented by processes. Communication is represented by signals that can take place between processes or between processes and the

environment of the system. Under pressure of the industry, the UML 2.0 has adopted some core features that are inspired by the SDL, such as a framework for dealing with Actions. The UML profile for Communicating Systems (UML-CS) [2] complements these extensions, to provide full support for SDL modeling, within the UML. For example, the UML-CS provides support for the definition of timers. Over the years, the industry has developed powerful code generators that take SDL models as input, map the models to some internal code representation and perform aggressive code optimizations. The generated code is mostly C code.

Typically, the process from requirement analysis to implementation and testing involves the following steps. The initial requirements are collected in a text document. System analysis results in UML models and Message Sequence Charts (MSC) [3] that capture the typical scenarios. The classes are mapped to SDL semi-automatically, and the SDL diagrams are completed manually to a level where they can be checked for consistency, simulated and validated. A test suite is generated from the SDL specification and finally, executable code is generated fully automatically.

Yet, the modeling process is time-consuming and error prone. All implementation details need to be reflected at the modeling level, before the application models can be simulated, and code can be generated. Many of these implementation details relate to the non-functional requirements of these applications, such as fault-tolerance, logging or security. In many cases, the hierarchical decomposition paradigm of SDL does not enable those concerns to be well modularized into separate SDL modules.

The Aspect-Oriented Software Development (AOSD) [4] community identifies those concerns as being crosscutting concerns. Their implementation cannot be well encapsulated within the modularity units of the language, because they follow different composition rules. Crosscutting concerns are pervasive in embedded software, because of the various constraints imposed by the environment and the stringent QoS requirements on the system. During the initial design phases, these concerns cannot be mapped from requirement to design in isolation, and end up tangled with model elements that implement other requirements.

This work proposes a framework for modularizing crosscutting concerns in embedded software, and automating their composition at different phases of the code generation process, including the modeling level, the different intermediary code representation levels and at the level of the final generated code. This paper discusses aspect modeling within the framework.

2 Approaches to AO Modeling

2.1 Lightweight UML Extension

The lightweight extension to the UML approaches take advantage of the UML extension mechanisms (stereotypes, tagged values and constraints) to refine the UML meta-model as to support AOP language constructs. The works of Pawlak [6], Aldawud [7] and Stein [8] fall into this category. These approaches provide graphical notations for aspects, for example, by defining pointcut associations, which link advices to join-points. The aspects tend to be tightly coupled to the core model. For large projects,

the relations might involve many classes of the application, and the visual representation of crosscutting does not scale well. The result of composition is complex and difficult to read and comprehend. The design model looks like a woven model.

The authors' position is that joinpoints are a validation and verification issue. This implies that during the design phase, we do not want to know where the joinpoints are, and hence, we do not need an explicit modeling construct for them.

2.2 Heavyweight UML Extension

The proponents of heavyweight extensions to the UML argue that aspects need to be first-class elements in the UML, because lightweight extension mechanisms are not expressive enough to capture weaving semantics by themselves. The heavyweight UML extension approaches define a self-contained metamodel that fits AOSD. The metamodel is obtained by tailoring the UML kernel metamodel. Heavyweight extensions provide graphical notations for aspects, but do not define the weaving concept explicitly. The works of Kandé [9], Han [10] and Lions [11] fall into this category.

Heavyweight approaches are complex to implement. Furthermore, when extending the UML metamodel, all the tools that realize test case generation, model validation and code generation need to be refactored to support the new AO metamodel.

2.3 Model Transformation Approaches

Model transformation approaches recognize the weaving process as a concept in the expanded metamodel of the UML. Model weaving approaches realize the coordination of crosscutting concerns with a base model through model transformation. Model weaving provides the capability to describe the essence of a concern into a separate model and transform other models accordingly. These approaches have the disadvantage that the AOP language constructs have no clear counterpart at the modeling level. The work of Gray [12] falls into this category.

2.4 Hybrid Approaches

Hybrid approaches to AOM recognize weaving as a concept in the modeling process, but leverage the weaving process by extending the UML metamodel. Typically, the metamodel for modeling core program elements is extended to support a join point model. The metamodel for modeling aspects is extended to support the notion of aspect, advice, pointcut and intertype declaration. A model weaver composes the aspect models with the core models by associating advice bodies to joinpoints, and intertype declaration to core model elements.

Tkatchenko [13] proposes a simple joinpoint model that is implemented through a heavyweight UML extension. The weaving process can partly be made implicit, because the extended metamodel extension captures part of the weaving semantics.

The work of Clarke and Baniassad [14] proposes a UML design model that encompasses different separation-of-concern techniques, including aspect-oriented de-

compositions. A Theme represents a modularized view of a requirement or concern in the system. The behavior of aspects is modeled using diagram templates. Themes are coordinated semi-automatically by an engine that applies merge and replace rules, and parameterizes the diagram templates, by binding advised methods to concrete joinpoints.

The work of Reddy [15] also takes advantage of UML templates for designing generic crosscutting features. Before composing the models the weaver instantiates the aspect models by binding application-specific elements to the template parameters.

2.5 AOM and MDA

The hybrid approach has interesting connections with the OMG Model Driven Architecture (MDA) [6]. The MDA separates platform independent details from the platform specific details of the system in a series of models. The approach is for each concern, to automatically expand more abstract diagrams by substituting the more concrete sub-diagrams into the abstract one. MDA proceeds by defining model transformation specifications that will automatically introduce, for example, platform specific behavior into a platform independent model. The transformation maps model elements of one domain metamodel to model elements of a target metamodel. The transformation is defined according to a transformation metamodel that defines the primitives of the transformation.

The hybrid approach defines a domain metamodel (a metamodel for modeling aspects), a target metamodel (a metamodel to visualize crosscutting behavior) and model transformation rules that conform to an AOSD metamodel. Aspects inject some more specific concern implementations into a more abstract model.

The relationship between MDA model transformations and model weaving is analogous to the relationship between AOP languages and Meta Object Protocol (MOP) languages. MOP's enable more powerful transformations, but are hard to write. AOP restricts the transformation space to a limited set of transformations, on a limited set of language elements. Aspects are less powerful, but provide better modularization units, and cover much of the MOP transformation space.

3 Aspect-Oriented Composition for Communicating Systems

3.1 Requirements

In an MDD setting, the SDL statechart weaver needs to accommodate the following constraints:

- The semantics of the SDL metamodel elements used to model the core application can not be modified to accommodate aspect weaving. It should be possible to simulate, validate and generate code from those models with the existing MDD tool suite.
- The aspect behavior needs to be specified in standard SDL, so that crosscutting behavior can be simulated and validated independently, and that code can be gen-

erated with the existing MDD tool suite. The metamodel used to model aspect behavior can therefore not break the semantics of SDL.

- It should be possible to validate and simulate the result of the composition between core models and aspect models using the existing MDD tool suite. The statechart weaver should therefore produce models that conform to the SDL metamodel.

Within an industrial setting and a specific domain, many crosscutting concerns tend to have common implementations from an application to another. The models that implement crosscutting concerns are themselves reusable from one application to another. What differs from one application to another are the bindings between aspect and core model. Therefore, we adopt an Aspect-Oriented architecture that involves a connector – a component that binds generic (within the domain) advices to application specific joinpoints. It encapsulates pointcut expressions and context binding definitions between joinpoints and named advices. The aspect advices are therefore made more reusable, as they do not depend on application-specific pointcut expressions.

This approach differs from the previous ones in that neither the core models, neither the aspect models require specific support for aspect-orientation. The author's position is that much of the complexity of aspect-oriented modeling is due to the difficulty of representing the aspect/core semantic interactions. This complexity can be overcome by defining separate diagrams that model the composition between the aspect behavioral diagrams and the core models.

3.2 AspectSDL: An SDL Statechart Weaver

The crosscutting behavior is modeled using regular SDL statecharts. Those models are referred to as SDL Aspect Beans. A connector defines the bindings between aspect bean statecharts and the core model statecharts. Figure 1 illustrates the architecture of the AspectSDL weaving engine. The SDL statechart weaver operates according to four distinct metamodels:

- A lightweight extension to the SDL profile that defines the joinpoint concept for SDL statecharts
- A lightweight extension to the UML metamodel that defines the concepts of aspect bean, advice, and intertype declaration.
- A connector metamodel that specifies how the bindings between aspect beans and model elements are defined. The connector metamodel is a structural weaving metamodel. It defines the static concepts the model weaver uses to identify cross-cutting, such as pointcut and advice declaration.
- A behavioral weaving metamodel that specifies how weaving strategies are defined. A weaving strategy defines the operations to be performed to bind an aspect bean element to the core model. Weaving strategies define the weaving semantics. The behavioral weaving metamodel defines a metaprogramming API for the weaving engine.

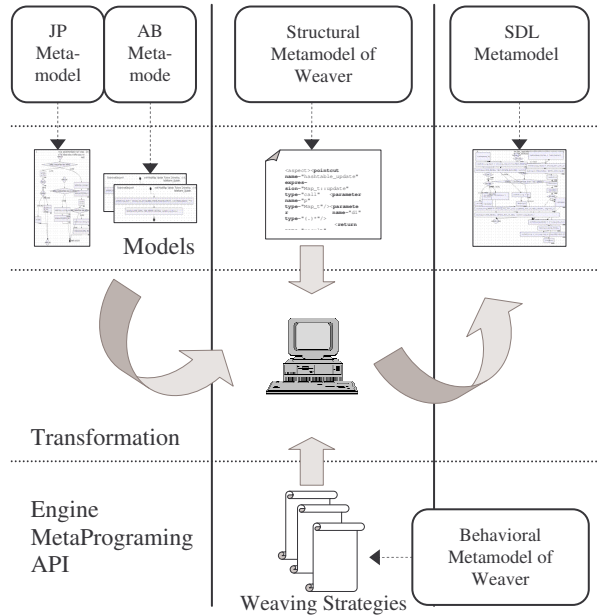


Fig. 1. Architecture of the AspectSDL weaving engine. The SDL statechart weaver operates according to four distinct metamodels: a joinpoint metamodel, an aspect bean metamodel, a connector metamodel and a behavioral metamodel of the weaver

3 AspectSDL Metamodels

4.1 Aspect Bean Profile

The Aspect Bean profile defines how the AspectSDL weaver looks at the Aspect Bean SDL statecharts. Aspect Beans are defined as Class stereotypes. We adopt an aspect model where advices have an explicit name. This allows advices to be defined independently of pointcut expressions. Advices are defined as Operation stereotypes. This is consistent with Stein's UML notation for Aspect-Oriented Design [8]. An advice is a special kind of operation that may not be called explicitly. InterType attributes are defined as Attribute stereotypes and InterType operations are defined as Operation stereotypes.

4.2 Joinpoint Profile

The extensions to the SDL metamodel described in this section are defined with respect to the UML profile for SDL [2], as represented in Telelogic TAU G2 [16].

The Joinpoint profile defines which elements in the SDL statecharts can be considered as joinpoints. The joinpoint metamodel should be as simple as possible. A simple joinpoint metamodel is one that allows various types of joinpoints to be treated uni-

sidered joinpoints. Yet, this metamodel captures elements that are irrelevant to aspect weaving, while it does not cover method return or method execution joinpoints.

Figure 3 depicts some of the joinpoints that are captured by the caller side joinpoint stereotype. Aside from the usual types of joinpoints such as method call (ExpressionAction), exception throwing (ThrowAction), and the return joinpoint (ReturnAction) it also supports new types of joinpoints that are useful for embedded and distributed software development, such as Set Timer, Reset Timer, Send Signal and Receive Signal.

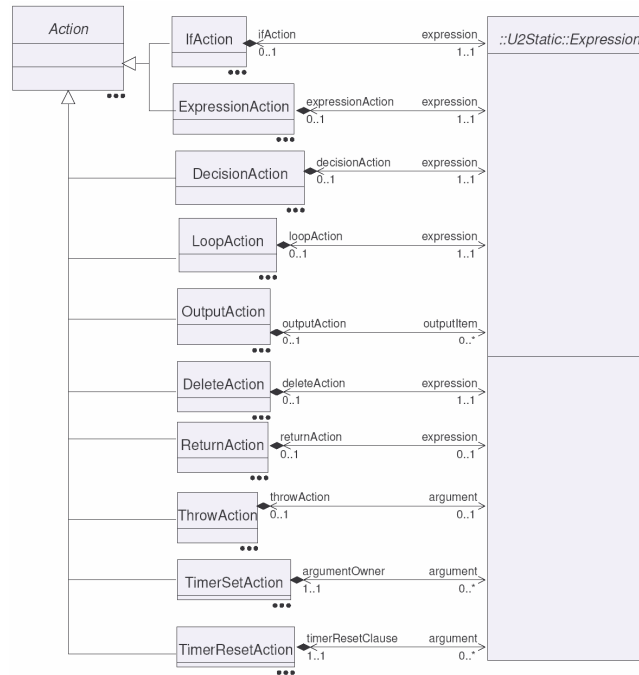


Fig. 3. Caller side joinpoint types

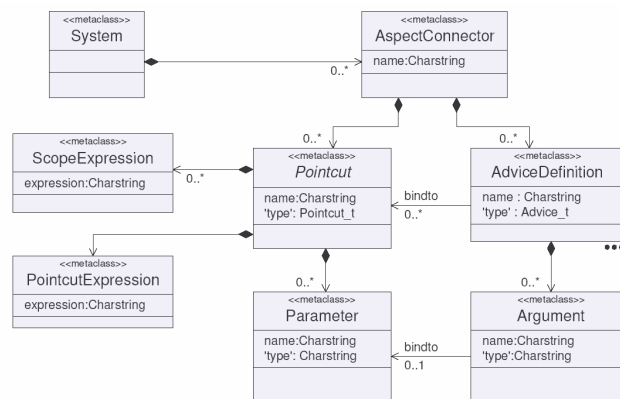


Fig. 4. The Connector Metamodel

4.3 Connector Metamodel

The connector is used in the first phase of the weaving process, to identify all active joinpoints in the core model, and the advices of the aspect bean. The connector also defines the binding between advice arguments and pointcut parameters. It captures context passing between the core model elements and the aspect bean model elements. The connector metamodel of Figure 4 defines the structure of the aspect-to-core binding definition. An Advice definition is bound to one or more pointcuts. A pointcut is composed of a pointcut expression and a scope expression which narrows the scope of the pointcut. Scope expressions include ‘within’ and ‘cflow’ expressions.

4.3 Weaver Behavioral Metamodel

SDL statechart weaving primary serves simulation and validation purposes. The goal of the AspectSDL framework is larger. The framework intends to provide pluggability of crosscutting concerns at different phases of the code generation process, including the modeling level, the different intermediary code representation levels and at the level of the final generated code. Weaving at the level of the intermediary code representation needs to be consistent with model weaving, as to not affect the simulation, verification and validation processes. There is therefore a need for mechanisms that provide traceability between the weaving semantics at the model level, the intermediary code representation level and the code level. The weaver behavioral metamodel intends to provide a common set of primitives to describe the weaving semantics at those different levels, using weaving strategies. The metamodel would enable the actions of the weavers of the framework to be consistent. At the second phase of the weaving process, joinpoints and advices have been identified, and their bindings are defined. Weaving strategies define the operations to be performed on the core model to implement the binding definition. These operations may vary from one type of joinpoint to another and from one type of diagram to another.

5 Conclusions

We distinguish between modeling crosscutting behavior (Aspect Beans) and modeling the aspect composition. The authors’ position is that the UML is suited for modeling aspect beans, but that it cannot accommodate the complex semantics of the weaving process. Therefore, we dedicate a separate metamodel for an aspect connector, whose role is to define primitives for modeling the composition between models that encapsulate the behavior of aspects and the models that capture the core concerns of the application. The paper illustrates this approach by presenting a model weaver for SDL statecharts. AspectSDL adopts an aspect modeling approach that recognizes weaving as a specific type of model transformation. The AspectSDL composes core models and aspect bean models according to a binding definition (connector) and weaving strategy definitions. Lightweight extension to the SDL profile are proposed, a connector metamodel is presented and a weaver behavioral metamodel is discussed.

Acknowledgements

This work is partially supported by CISE NSF grant No. 0137743, and performed at Motorola Labs.

References

1. ITU, Z. 100: Specification and Description Language (SDL), ITU-T, Geneva (2000)
2. ETSI: UML Profile for Communicating Systems, DTR/MTS-00085 (2004)
3. ITU, Z.120: Message Sequence Charts (MSC), ITU-T, Geneva (2000)
4. Kiczales, G., *et Al.*: Aspect-Oriented Programming. Proceedings of the European Conference on Object-Oriented Programming, Springer-Verlag (1997)
5. OMG: Model-Driven Architecture homepage <http://www.omg.org/mda/> (2000)
6. Pawlak, R., *et Al.*: A UML Notation for Aspect-Oriented Software Design. 1st International Workshop on Aspect Oriented Modeling at the 1st International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands (2002)
7. Aldawud, O., Elrad, T., Bader, A.: A UML Profile for Aspect- Oriented Software Design. 3rd International Workshop on Aspect Oriented Modeling at the 2nd International Conference on Aspect- Oriented Software Development, Boston, USA (2003)
8. Stein, D., Hanenberg, S., Unland, R.: A UML-Based Aspect-Oriented Design Notation for AspectJ. Proceedings of the 1st international conference on Aspect-Oriented Software Development, Enschede, The Netherlands (2002)
9. Kandé, M.M., Kienzle, J.,Strohmeier, A.: From AOP to UML , A Bottom-Up Approach, Aspect-Oriented Modeling with UML workshop at the 1st International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands (2002)
10. Han, Y., Kniesel G., Cremers A.: Towards Visual AspectJ by a Meta Model and Modeling Notation, 6th International Workshop on Aspect-Oriented Modeling at the 4th International Conference on Aspect-Oriented Software Development, Chicago, USA (2004)
11. Lions, J.M., Simoneau, D., Pilette, G., Moussa, I.: Extending OpenTool/UML Using Meta-modeling : An aspect-oriented programming case study, 2nd International Workshop on Aspect Oriented Modeling, UML 2002, Dresden, Germany (2002)
12. Gray, J.: Aspect-Oriented Domain-Specific Modeling: A Generative Approach Using a Meta-weaver Framework, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville (2002)
13. Tkatchenko, M., Kiczales, G.: Uniform Support for Modeling Crosscutting Structure, 6th International Workshop on Aspect-Oriented Modeling at the 4th International Conference on Aspect-Oriented Software Development, Chicago, USA (2004)
14. Clarke, S., Baniassad, E.: Aspect-Oriented Analysis and Design. The Theme Approach Addison-Wesley, Object Technology Series, ISBN: 0-321-24674-8 (2005)
15. Reddy, R., France, R., Georg, G.: An Aspect-Oriented Modeling Approach to Analyzing Dependability Features, 6th Workshop on Aspect-Oriented Modeling at the 4th International Conference on Aspect-Oriented Software Development, Chicago, USA (2002)
16. Telelogic: TAU G2 homepage, <http://www.telelogic.com/products/tau/index.cfm> (2005)