Modeling Crosscutting Concerns in Reactive Systems with Aspect-Orientation

Mark Mahoney

mmahoney@carthage.edu

Abstract. This paper describes the research I am currently performing as a PhD student at the Illinois Institute of Technology with my advisor Dr. Tzilla Elrad. The focus of my research is Aspect-Oriented Modeling and reactive systems. In particular I am interested in the modeling of reactive objects using statecharts and scenarios.

1 Introduction

The goal of Aspect-Oriented Software Development (AOSD) is to separate crosscutting concerns from core concerns. The focus of my research is to tackle one of the most important parts of Aspect-Oriented software design- modeling. Currently, the modeling techniques that address crosscutting concerns in reactive systems lack the expressiveness to model interactions between core and crosscutting concerns effectively. In particular, my research focuses on modeling class behavior with statecharts and scenarios. In the course of my research I have developed a methodology and framework that allows two or more independent statecharts, one for a core concern and one or more for crosscutting concerns, to be composed into a single model. The statecharts interact by having their events take on new meaning in the composed model. A developer specifies the new meanings declaratively. Concerns can then be modeled in isolation and brought together only when necessary.

The goals of my future work are to further examine the use of statecharts and to explore scenario based languages for addressing crosscutting concerns. With respect to the statechart work I have begun, I am looking into other contexts such as ubiquitous computing and how crosscutting concerns are handled in those environments. On a parallel track, I intend to continue exploring Aspect-Oriented Modeling by researching techniques to model crosscutting concerns in scenario based languages like Live Sequence Charts (LSC)[6]. LSC's allow one to model user concerns in a formal language similar to UML use cases and sequence diagrams. Tools have been developed to verify these models are consistent and complete. I will be exploring the use of Aspect-Oriented use case research and scenarios.

2 Aspect-Oriented Software Development and Modeling

One of the limitations of object-oriented software development is that it does not have an efficient way of expressing crosscutting concerns. A crosscutting concern is one that cannot easily be modularized into a single unit such as a class. Rather, it is spread out or tangled with the implementation of other concerns.

For example, synchronization is a concern that developers deal with whenever guarded access to a resource is required. Access to devices, files and memory are all things that may require synchronization. The idea of synchronization of a resource inside a class is common, yet there is no easy way to express this idea and encapsulate it in its own first class entity for reuse in other applications. Rather, anytime synchronization is required for an object it is injected in a core concern's class. The same code may be repeated in several different places, once for each class that requires some form of synchronization. The tangled code implementing the synchronization interferes with the logical flow of the core code where it is injected. Further, the full synchronization code may be scattered across many units. A tangled, scattered implementation complicates the reuse of both the core and the synchronization code. Aspect-Oriented Software Development provides a mechanism to encapsulate crosscutting concerns into logical units, called aspects, and weaves them with the core components to form a working system.

3 Behavioral Modeling with Statecharts

Statecharts [5] are a tool used to model the reactive behavior of a class. A statechart is an effective design tool when the decision on what action to take in response to a given input is dependent not only on the input, but also on the current state of the system. An object's behavior can be decomposed into states. A state captures a system's history while abstracting away unhandled outside stimuli. Activities may take place while in a state or on a transition between states. An event handled in one state may cause a completely different reaction than the same event in another state. A statechart specifies three things: the stable states in which an object may live, the events that trigger a transition from state to state, and the actions that occur on each state change [11].

A simple statechart is said to have an 'exclusive-or' relationship between the states. This 'Or-composition' means that a statechart may be in one, and only one, state at a time. A statechart may also exhibit 'And-composition' by employing orthogonal regions. A statechart with orthogonal regions is composed of two or more independent statecharts that run concurrently and broadcast their events. Being in a statechart with orthogonal regions means that in every region, one and only one, of the states from each composed statechart will be active. Orthogonal regions allow you to avoid intermixing the independent behaviors as a cross product and, instead, keep them separate [11].

Currently, the Unified Modeling Language (UML) [10][1] is the language of choice when modeling reactive systems using statecharts. In a recent work [2], the authors have proposed an aspect-oriented design technique that uses UML/Statecharts to simplify the design of object-oriented software systems. The approach provides developers with a methodology to create semi-independent statecharts that broadcast events to orthogonal regions that represent core and aspect behaviors.

The statecharts in [2] describe the dynamic behavior of separate concerns. Each of the statecharts exist together in orthogonal regions and communicate through broadcast events. The broadcast events are used as the mechanism for implicit weaving of aspect and core statecharts.

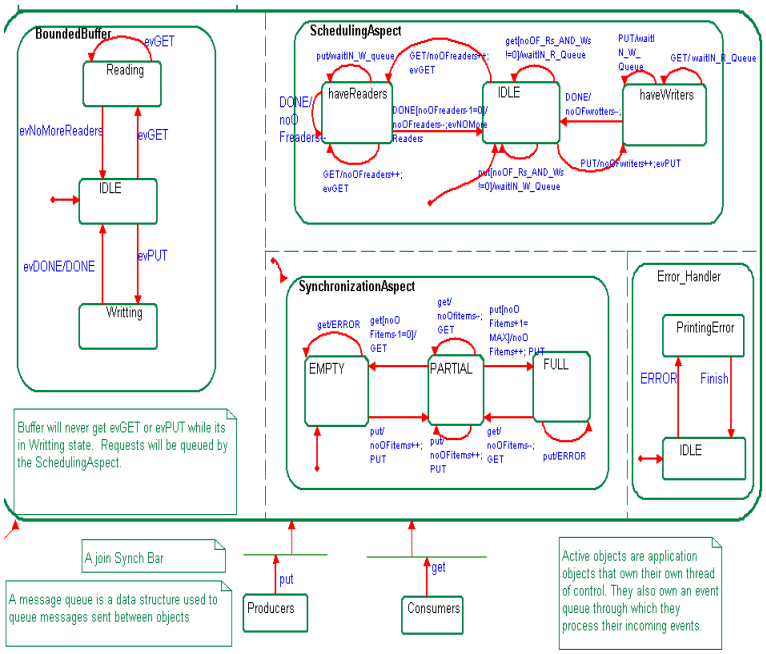

Figure 2

This approach requires that the developer hard code the propagation of events from core statechart to crosscutting statechart. In figure 2 a bounded buffer (core concern) is guarded by synchronization and a scheduling statechart (crosscutting concerns). When a 'put' event is introduced while in the EMPTY state a transition will occur in the synchronization statechart and a new event, 'PUT', will be broadcast. This new event will be handled in the scheduling statechart. There is a lack of obliviousness in [2] resulting in the core developer and the aspect developer being aware of each other's state transitions and events.

My approach [8] extends the ideas in [2] by replacing hard coded event propagations with high-level declarations about how an event in one statechart can be treated like a completely different event in another statechart. That is, in [2] there is no attempt reduce dependencies between statecharts, each statechart is highly dependent on the broadcast events in other orthogonal regions. The difference in my work is to attempt to reduce the dependencies between statecharts in different orthogonal regions by allowing broadcast events to be reinterpreted to take on different meanings in different orthogonal regions. The key contribution of my approach is to make the individual statecharts more reusable while continuing to keep concerns separate.

An Aspect-Oriented Statechart Framework (AOSF) was created to implement these ideas as a proof of concept [13]. With the framework, a developer can take two or more independently developed executable statechart models and use the framework classes to create an executable version of those statecharts. Further, a developer can weave them together with some high level declarations about how the statecharts crosscut each other and which events shall be reinterpreted in the orthogonal regions. This can be done without hard coding events in the submodels.

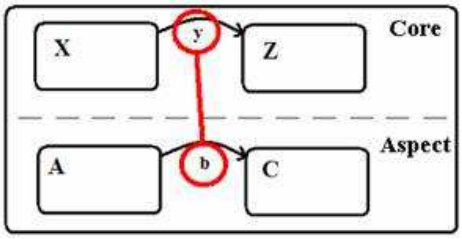

Figure 3

The semantics of this model can best be stated as:
If the core statechart is in State 'X' and event 'y' is introduced, and if the aspect statechart is in State 'A', treat 'y' exactly as if it were event 'b'. The aspect statechart can handle the event before or after the core statechart.

I have been able to apply these concepts to address different types of crosscutting concerns. Model Driven Architecture (MDA)[7] is a new technology that attempts to separate the platform independent details from the platform specific details of a system in a series of models. The models are then used to generate code. The idea is that platforms change relatively frequently (programming language, programming paradigm, hardware, operating system, etc.) but the core business models change infrequently. The developer devotes his efforts to developing Platform Independent Models (PIM) that persist for long periods of time. When a platform change is required all that needs to be done is to create a new Platform Specific Model (PSM) to handle the differences in platform, the core business models remain unchanged.

One particular type of model is a Virtual Finite State Machine (VFSM) [11]. VFSMs, like MDA implementations, separate platform independent behavior from platform

specific behavior. The platform independent behavior is modeled in an extension to Extended Finite State Machines (EFSM) and the platform specific behavior is modeled in the form of input processor and output processor classes that interact with the VFSM. There is a high degree of coupling between the state machine and the input/output processors. Our research [9] has argued that platform specific details in a VFSM are crosscutting concerns and can be modeled using Aspect-Oriented Modeling techniques including my Aspect-Oriented Statechart Framework [8] and VFSMs.

It is my belief that platform specific models (input and output processors) are not the core concerns in a VFSM implementation. The input and output processors are crosscutting concerns that don't need to be tangled with core VFSM state machine. In the spirit of MDA my work seeks to separate platform independent characteristics from platform specific characteristics of a system and weave them together as needed. I propose using the AOSF to model the input and output processors as well as the VFSM as statecharts that crosscut each other. Then the statecharts can be woven together and the events from the input statechart can be reinterpreted to have meaning in the core VFSM. Further, events in the core VFSM that are related to virtual actions can be reinterpreted to have meaning in the output processor statechart.

The main benefit of this approach is that there is little direct coupling between the input/output processors and the VFSM. In fact, any statechart created using the framework can be used as an input/output processor. All that needs to be done is to weave the statecharts together and map events from the input processor to the VFSM and from the VFSM to the output processor. This increases the likelihood of achieving reuse from a statechart library.

4 Future Work towards the PhD

I plan to continue working with state based systems. I am currently researching aspects as they apply to ubiquitous computing and variability. I am interested in discovering if statecharts controlling separate machines may be combined into one statechart with orthogonal regions for the purpose of coordination and control. These distributed statecharts would travel to different orthogonal regions of different machines based on the location of the device. For example, an airplane may have a statechart that describes its current state (ascending, descending, etc). Each air traffic control tower across the country may require that as a plane flies through its airspace it add its statechart to the tower's statechart in an orthogonal region. The tower can then reinterpret events from each plane to control the airspace. For example, an air traffic control tower in Washington D. C. may reinterpret a 'descent' event as a threat, whereas that same event may be reinterpreted to an indication of landing in another tower's statechart. I intend on altering the AOSF to handle distribution and movement of statecharts.

I have just begun to research the effect of crosscutting concerns on scenario-based languages. A scenario, like a use case, models a typical interaction a user has with a system to provide some useful result. One particular scenario based language, called Live Sequence Charts (LSC)[6], deals with reactive systems. Scenario-based descriptions of behavior are not immune to crosscutting concerns. I am researching ways to address crosscutting concerns in scenarios by proposing some additions to the LSC language and an accompanying tool called the Play Engine. The proposals I have made would allow one to *see* the effect of applying crosscutting concerns to different scenarios. I have only recently begun this line of work but I believe there is great potential to incorporate recent research into aspect-orientation and use cases to these scenarios.

References

1. Booch, Rumbaugh, Jacobson. 1999. The Unified Modeling Language User Guide. Addison Wesley.
2. Elrad T., Aldawud O., Bader A.. "Aspect-oriented Modeling - Bridging the Gap Between Design and Implementation". Proceedings of the First ACM SIGPLAN/SIGSOFT International Conference on Generative Programming and Component Engineering (GPCE). Pittsburgh, PA. October 6–8, 2002, pp. 189-202.
3. Filman, R. Friedman, D. Aspect-Oriented Programming is Quantification and Obliviousness, Workshop on Advanced Separation of Concerns, OOPSLA 2000, October 2000, Minneapolis. http://ic-www.arc.nasa.gov/ic/ darwin/oif/leo/filman/text/oif/aop-is.pdf
4. Flora-holmquist, A., Staskauskas, M. "Formal Validation of Virtual Finite State Machines" Proceedings of the workshop on Industrial-Strength Formal Specification Techniques 1995, pp 122-129.
5. Harel, David. 1987. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming 8 231-274.
6. Harel, D, Marelly, R. 2003. Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer-Verlag
7. Kleppe, A., Warmer, J., Bast, W. "MDA Explained The Model Driven Architecture: Practice and Promise" Addison-Wesley, 2003.
8. Mahoney, M., Bader, A., Aldawud, O., Elrad, T., "Using Aspects to Abstract and Modularize Statecharts" The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004. http://www.cs.iit.edu/~oaldawud/AOM/mahoney.pdf
9. Mahoney, M., Elrad, T. Modeling Platform Specific Attributes of a System as Crosscutting Concerns using Aspect-Oriented Statecharts and Virtual Finite State Machines, the 6th International Workshop on Aspect-Oriented Modeling as part of AOSD'05 (Chicago, USA, March 2005) http://dawis.informatik.uni-essen.de/events/AOM_AOSD2005/Mahoney.pdf
10. Object Management Group, Unified Modeling Language Specifications. Version 1.5 Mar 2003.
11. Samek, Miro. 2002. Practical Statecharts in C/C++. CMP Books.
12. Wagner, F. "VFSM Executable Specification" on CompEuro 1992, Hague, Holland.
13. AOSF Home Page. http://ulysses.carthage.edu/faculty/mmahoney/AOP/AOSF/default.htm