TouchRAM: A Multitouch-Enabled Tool for Aspect-Oriented Software Design

Wisam Al Abed, Valentin Bonnet, Matthias Schöttle, Engin Yildirim, Omar Alam and Jörg Kienzle

School of Computer Science, McGill University, Montreal, QC H3A E09, Canada
Omar.Alam@mail.mcgill.ca, Joerg.Kienzle@mcgill.ca

Abstract This paper presents TouchRAM, a multitouch-enabled tool for agile software design modeling aimed at developing scalable and reusable software design models. The tool gives the designer access to a vast library of reusable design models encoding essential recurring design concerns. It exploits model interfaces and aspect-oriented model weaving techniques as defined by the Reusable Aspect Models (RAM) approach to enable the designer to rapidly apply reusable design concerns within the design model of the software under development. The paper highlights the user interface features of the tool specifically designed for ease of use, reuse and agility (multiple ways of input, tool-assisted reuse, multitouch), gives an overview of the library of reusable design models available to the user, and points out how the current state-of-the-art in model weaving had to be extended to support seamless model reuse.

1 Introduction

Model-Driven Engineering (MDE) [12] is a unified conceptual framework in which the whole software life cycle is seen as a process of *model production*, *refinement,* and *integration*. High-level specification models are refined or combined with other models using *model transformations* to include more and more solution details and to ultimately produce a model that can be executed.

In practice, MDE faces several important challenges that prevent the widespread adoption of modeling as a means to improving the software development process. In the context of this work, the two relevant challenges are *scalability* and *reusability* of models. Models of complex applications tend to grow in size, to a point where even individual views are not readily understood or analyzable anymore. Furthermore, building complex models is very time consuming: models are often created from scratch, as opposed to reusing existing models.

Aspect-orientation modeling (AOM) techniques define special kinds of model transformations called *model weavers* that have been successfully used to separate and compose crosscutting concerns within software models, focussing in particular on the intricacies of concern interactions and conflicts. AOM makes it possible to package models of generic concerns in such a way that they are easy to reuse within other models. Furthermore, by providing weaver support for model hierarchies, complex models can be build by composing existing ones.

This paper presents TouchRAM, a multitouch-enabled tool for agile software design modeling aimed at developing scalable and reusable software design models. The tool gives the designer access to a vast library of reusable design models

encoding essential recurring design concerns and provides support to rapidly apply these concerns within the design of the software under development. This is enabled by exploiting model interfaces and aspect-oriented model weaving techniques as defined by the Reusable Aspect Models (RAM) approach [10].

The paper is structured as follows: Section 2 gives an overview of the tool implementation and background on RAM. Section 3 highlights the tool features specifically targeted at agile software design modeling: subsection 3.1 presents how the GUI streamlines model manipulation; subsection 3.2 outlines the library of reusable design concern models available to the user and explains how they can be applied within an application model, and subsection 3.3 explains how the tool supports working with model hierarchies. Section 4 points out how the current state-of-the-art in model weaving had to be extended to support seamless model reuse and model hierarchies, and the last section draws some conclusions.

2 Background

This section presents background information on the tool, i.e., what technologies and frameworks it is built on and other implementation details. In order to make the tool accessible to a wide audience, cross-platform compatibility was one of the major design concerns. We therefore opted to do our development entirely in Java. As a result, all libraries and frameworks we considered to use to build our user interface and model transformation backend had to be implemented in Java as well.

2.1 Architecture

TouchRAM consists of the front end, i.e., the graphical user interface (GUI) and the backend, which contains the RAM meta-model and the RAM model weaver.

The GUI of the TouchRAM tool is realized using the open source Java framework *Multitouch for Java* (MT4j) [3]. MT4j is a framework for creating visual applications in 2D or 3D using OpenGL for software or hardware accelerated graphics rendering. An event stack that allows for different kinds of input events is also provided; the tool ships with support of mouse and keyboard input as well as multitouch input through the TUIO protocol [4]. TouchRAM has been tested for 32 and 64 bit architectures on the Mac OSX, Windows and Linux (Ubuntu) platforms, however, its dependence on Java and TUIO means it should work on any environment where both these are supported.

While the user interface of TouchRAM relies on MT4j, all other components of the tool are decoupled from the GUI based on a Model-View-Controller design. This makes separate evolution of the GUI and the backend possible.

The foundation of the backend is the RAM meta-model that defines the abstract syntax for RAM models created with the tool. The meta-model is defined using the Eclipse Modeling Framework (EMF) [14]. The provided facility allows us to define the structured data model and generate the required Java code that is used by TouchRAM. Furthermore, we are able to serialize our models in XMI (XML Metadata Interchange) format corresponding to that meta-model. Command-based editing provided by EMF.Edit is used in order to offer the user undo/redo functionality. The User Interface is notified of changes to the model

through EMFs built-in notification mechanism. The Object Constraint Language (OCL) is used for constraints on the meta-model, specification of derived properties and implementation of operations defined in the meta-model.

The RAM weaver, which is invoked by the GUI on command of the user, is capable of composing multiple models together. Instead of directly implementing the weaving with Java, TouchRAM uses the Kermeta workbench [1]. Kermeta provides an object-oriented model transformation language based on lambda expressions similar to OCL and works with EMF-based meta-models. Furthermore, Kermeta includes support for aspect-orientation. Transformations written in Kermeta are compiled into Scala code, which runs on a standard Java VM.

2.2 Reusable Aspect Models

TouchRAM is based on Reusable Aspect Models (RAM), an aspect-oriented multi-view modeling approach that integrates class diagram, sequence diagram and state diagram AOM techniques [10]. As a result, RAM aspect models can describe the structure and the behavior of a concern under study. Currently, however, TouchRAM only supports structural modeling.

RAM aspect models define an aspect interface that clearly designate the functionality provided by the aspect, as well as its mandatory instantiation parameters [5]. When an aspect model is applied, all mandatory instantiation parameters must be mapped to compatible model elements in the application model. Flexibility is achieved by allowing any model element to optionally be composed or extended. RAM supports the creation of elaborate aspect dependency chains. This makes it possible to model an aspect that provides complex functionality by decomposing it into aspects that provide simpler functionality. At the same time, aspects providing simpler functionality can be reused in several aspects of complex functionality. As a result, scattering and tangling of models can be prevented at all complexity levels.

3 Tool Features supporting Agile Software Design

Modeling raises the level of abstraction in comparison to source code, and therefore has the potential for enabling fast exploration of software designs. To make this possible, though, a modeling tool must be designed accordingly. In this section we report on three key features of the TouchRAM tool specifically targeted at agile software design: the streamlined model manipulation capabilities offered by the TouchRAM GUI, the reusable design concern library, and navigation through different levels of abstraction.

3.1 Streamlined Model Manipulation

Exploiting Platform Capabilities The GUI of TouchRAM has been designed to provide intuitive and fast model manipulation capabilities to ensure that the user can focus entirely on the task of modeling. This starts by maximally exploiting the input and output hardware of the platform that the tool is running on. On the input side, TouchRAM supports multitouch and gesture-based input as well as standard mouse and keyboard. The tool was designed without modes, i.e., at any time, the modeler can use gestures or the mouse/keyboard to manipulate the model, depending on what input is most efficient. On the output side,

TouchRAM is designed to support heterogeneous screen sizes, mainly by providing high performance support for panning and zooming. The user can change the zoom level at any time by either using the mouse wheel or a two-finger scale gesture. To assure that a model created on one screen can be opened on a different screen without losing the overall overview of the model, TouchRAM scales models according to the current screen dimensions when opening.

Intuitive Editing General tasks are performed using simple gestures (i.e., tap, double-tap and tap-and-hold) that work with both mouse and touch input. For example, tap-and-hold is used to create or edit model elements. When performed on the background, a class is created. Tapping-and-holding on a class puts the class in edit mode which allows to delete or add attributes or operations. Double-tapping allows to edit an element or one of its properties. When done on the visibility field or return type of an operation or the type of an attribute, a selector menu pops up displaying semantically correct choices for that element.

Certain manipulations can be done very efficiently using multitouch. For instance, the tool can recognize advanced gesture commands, i.e., drawing a rectangle to create a class, performing a zig-zag movement to delete a model element, or drawing a line to create an association. For users that do not have access to multitouch input, TouchRAM offers (less efficient) mouse equivalents for these commands, for instance, double-clicking on one class to put it into edit mode, and then double-clicking another class to create an association between the two or clicking and holding to inherit from that class.

Of course there are also manipulations that are more efficiently done with the keyboard and mouse, e.g., writing text or detailed positioning of model elements. For users that do not have access to a keyboard, TouchRAM displays a popup touch-keyboard whenever a text input is expected.

Some manipulations can also be accomplished in multiple ways. For instance, when adding a new operation to a class and a keyboard is available, the whole signature can be written at once. The given signature is then parsed and checked for conformance to the meta-model. For example, entering "+ String getName()" creates an operation called getName which is public, has no parameters and returns a String. If no keyboard is available, only the model element names need to be provided using the touch-keyboard. The tool displays the potential semantically valid visibility, parameters and return types choices to the modeler, who selects the desired elements with a simple tap.

Tool-assisted Layout Currently TouchRAM does not provide support for automated layout of class diagrams. However, the modeler can rearrange classes one by one, or multiple classes simultaneously using multitouch gestures. The relationships between classes, i.e., associations and inheritance, are repositioned automatically by the tool using a sophisticated algorithm. The relationship lines are attached to the class angle depending on the relative position of the related classes, minimizing line crossings.

3.2 Library of Reusable Design Concern Models

The success of modern programming languages such as Java is partially due to the fact that they ship with a significant library of reusable code. The Java Class

Library as an example offers a programmer thousands of classes that provide solutions for common implementation concerns, including classes for all common data structures, such as lists, trees, and maps. That way, a programmer does not need to code these classes herself, but simply reuses their behavior by instantiating them and calling the appropriate methods.

The idea of the reusable design concern model library (RDCML) that ships with TouchRAM is similar, but is applied to modeling. Its purpose is to increase modeling productivity by providing models for common design concerns that a modeler can use within an application model with minimal effort when appropriate[1]. Each model in the library is self-contained, i.e., it contains all the structural and behavioral model elements pertaining to a design concern, and is typically relatively small (1 - 6 classes). The *model interface* clearly designates all the classes, associations and operations that are visible, i.e., that can be instantiated / called at runtime to invoke the functionality provided by the model. The current models in the RDCML are organized into the following categories:

- The *design patterns* category contains aspects for the basic structural, behavioral and creational design patterns (e.g., *Singleton*, *Observer*, *Command*, etc.)
- The *utility* category contains aspects that provide basic functionalities like copying *(Copyable aspect)* and naming *(Named aspect)*, as well as data structures involving multiple objects, such as *Map*.
- The *networking* category contains aspects relevant to networking, such as *Serializer*, *SocketCommunication* and *NetworkedCommand*.
- The *workflow* category contains aspects that are useful whenever the application needs to define and execute flexible workflows. For example, the current library supports sequential, conditional, timed, nested and parallel execution of activities.
- The *transactions* category contains aspects that provide state checkpointing and recovery support, as well as design solutions for isolating concurrently running activities.

Applying a Reusable Design Concern When elaborating an application model, a modeler typically starts by defining application-specific structure and behavior. When appropriate, she can also choose to apply aspect models from the RDCML to complete her design. In RAM terminology this is called *instantiation*. Aspect models are reusable, because they can be instantiated multiple times in the same application model, or in several distinct application models.

When the modeler applies an aspect from the library, TouchRAM displays an instantiation view, which allows the modeler to establish a mapping from classes and methods in the library model (also called the lower-level aspect) to the application model (also called the higher-level aspect). The instantiation

[1] The RDML is not meant to replace standard class libraries. On the contrary, in order to access functionality provided by standard programming language libraries, TouchRAM currently provides support for importing Java classes into design models using reflection.

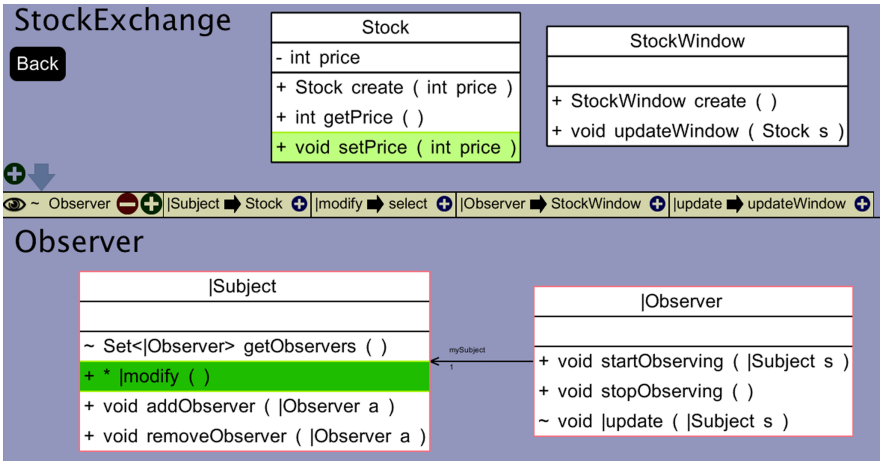

Figure 1. Instantiation View of TouchRAM

view is divided into two parts: the higher-level aspect is viewed in the top and the lower-level aspect is viewed in the bottom of the view. Tapping on a class in the bottom view highlights the classes that it can be mapped to in the higher-level aspect. Just like for standard model manipulations, the tool assists the modeler in creating semantically correct mappings. This is illustrated in Fig. 1. It depicts a situation where a modeler applies the *Observer* design pattern model to a *StockExchange* application model. She has already mapped the */Subject* class of the lower-level aspect to the *Stock* class, and the */Observer* class to the *StockWindow* class, and is now in the process of mapping the */modify* operation of */Subject*. Since */modify* is part of class */Subject* in the lower-level aspect, and */Subject* was already mapped to *Stock* in the higher-level aspect, TouchRAM marks only the methods of *Stock* with matching parameters as selectable.

The *Observer* example also nicely illustrates why reusable design concern models are more powerful than OO programming language libraries. Class libraries can only encapsulate implementation concerns if the structural and behavioral interface for the concern is contained in a single class. This is not the case for the *Observer* design pattern, since it involves two classes (*Subject* and *Observer*) with distinct behavioral responsibilities (*modify*, *notify*, *update*, etc.).

3.3 Navigating Levels of Abstraction

Instantiations can not only be used to reuse models of the RDCML. A modeler can define her own reusable aspect models, which allows her to decompose a big application model into several smaller inter-dependent models that describe different application-specific design concerns. TouchRAM supports complex model dependency chains, and hence big models can be built by combining many small aspect models that describe the design at different levels of abstraction. For example, the low-level aspect *Observer* shown in Fig. 1 actually depends on an even lower-level aspect called *ZeroToManyAssociation*, which uses the Java implementation class `java.util.Set` to link an instance of a */Data* class to many instances of the */Associated* class. The *Observer* aspect uses this low-level de-

sign concern to link a */Subject* instance with many */Observers*, as shown in the instantiation directives on the bottom line of Fig. 1.

When using TouchRAM, it is not uncommon to create software designs with model hierarchies with many layers of abstraction/models. A modeler can proceed in a top-down manner, starting at high-level application-specific models, and incrementally adding lower-level design details, either self-modeled or from the RDCML. Conversely, the modeler can also start by designing lower-level models first, and then raise their level of abstraction by adding higher-level models that depend on them.

TouchRAM allows the modeler to easily navigate through the model hierarchy. When she opens an aspect model, a list of its instantiations is shown in the bottom of the editing view. She can view the instantiated aspect simply by tapping on the eye icon of an instantiation: a new view opens to display the instantiated aspect. This allows the modeler to focus on each individual design concern at each level of abstraction in isolation.

TouchRAM also allows the modeler to visualize how a higher-level aspect interacts with a lower-level aspect. Tap-and-hold on an instantiation instructs the RAM weaver to combine the lower-level aspect with the higher-level one according to the instantiation mapping to yield a woven model that displays the model elements from both models, i.e., from both levels of abstraction. Using this feature, the modeler can selectively visualize specific lower-level design details, for instance for analysis reasons.

To experiment with different designs, the modeler can exchange design models at a given level of abstraction with other ones providing similar functionality. Typically this simply involves replacing an instantiation in the higher-level model, and then asking the weaver to compose the models again.

Finally, with the "weave all" command, the modeler can instruct the weaver to "flatten all levels of abstraction" and produce a woven model that contains all the specified design details. The details on how the weaver handles model hierarchies are presented in the next section.

4 Hierarchical Model Weaving

The core of the class diagram weaver in TouchRAM is based on the symmetric composition technique proposed by France et al. [11] that was implemented in a tool called *Kompose* [8,2]. In essence, Kompose merges two class diagrams into one by looking at the signature of the model elements in each diagram, and then combining those with matching signatures. After the merge, post-merge directives can be used in order to make changes to the resulting model, if necessary. In order to support reuse, pre-merge directives can be used to prepare a generic model for a specific use, e.g., to change general model element names to names used in the application model so that the signature-based weaving algorithm will merge them.

Kompose does not directly support aspect hierarchies as defined in RAM, and therefore can not be used as such to support incremental top-down or bottom-up modeling as described in subsection 3.3. We had to considerably modify and extend the approach to fit our needs.

4.1 Instantiation Types

To use a design concern model within another design model, the modeler specifies an instantiation mapping between the two models as described in subsection 3.2. Elements that can be mapped are classes, operations, associations, attributes, and parameters. Currently there are two types of instantiations that can be created: *depends* and *extends*. The two types correspond to the two different ways of using TouchRAM to incrementally build models of significant size.

The *depends* instantiation is used when the two models are modeling different levels of abstraction of the software design, as it is the case for top-down or bottom-up development. With *depends*, the modeler is required to provide mappings for all lower-level model elements that she wants to expose at the higher level. The visibility of all unmapped elements is by default switched from *public* to *aspect-private* [5] by the weaver to encapsulate the low-level details.

The *extends* instantiation is used when the designer's intent is to increment a current design model with additional functionality. Since in this case both design models are at the same level of abstraction, they often refer to the same model elements. Therefore, with *extends*, default mappings are created for all model elements that have the same signature, and all model elements from the lower-level model maintain their visibility properties during the weaving process.

4.2 Weaving Instantiations

In order to allow design exploration across levels of abstractions and increments in a flexible and agile way, our weaver must be capable of weaving any two directly dependent design concern models within a hierarchy together. The resulting model must be one that correctly replaces the two original models within the hierarchy. This is achieved by updating the instantiation directives from the lower level.

For example, in Fig. 2 A depends on B, which in turn depends on C and D. When weaving B into A to yield a new model A+B, our algorithm needs to update the instantiations made in B that originally mapped elements from C and D to elements in B to now map the elements from C and D to the corresponding elements in A. In our example, aspect A mapped */B->/A* and *Y->X*, but aspect B mapped */C->/B* and */D->Y*. After weaving B into A, the updated mappings are */C->/A* and */D->X*.

The general rule to update instantiations at weave time is as follows: Given two aspects A and B where A depends on B, for each mapping m1 in A where a left hand side element appears on the right hand side of a mapping m2 in B (text colored in red linked by dotted lines in Fig. 2), create a new mapping in A between the left hand side element of m2 and the right hand side elements of m1 (text colored in blue linked by dashed lines in Fig. 2).

4.3 Weaving Algorithm

The following list summarizes the steps that our weaver executes to weave a reusable design concern model B into model A:

1. Process *extends* instantiations: If the instantiation is of type *extends*, create default mappings for all model elements in B that have corresponding

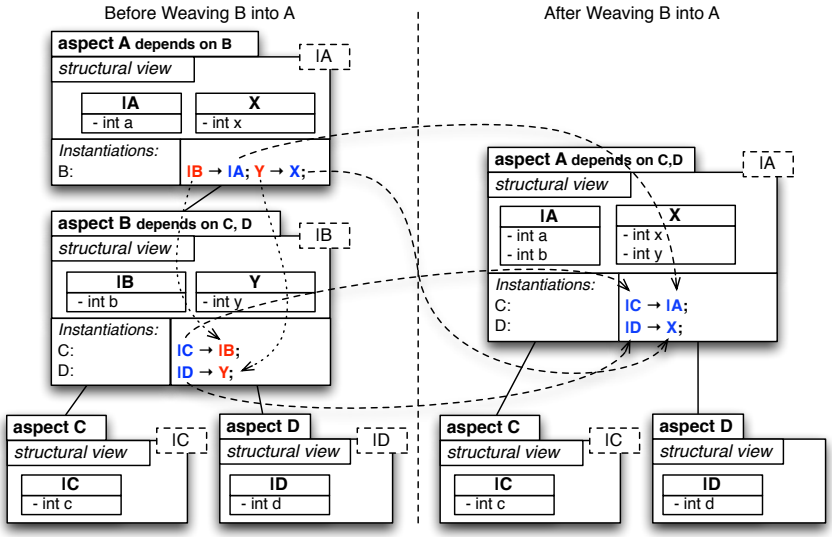

Figure 2. Updating Instantiation Directives During Weaving

model elements in A. For classes, the names must match. For operations, the signature must match.

2. Check for name clashes: In the woven model, two classes representing different design classes can not have the same name. Therefore, if at this point there exists a class in B with a name that matches the name of a class in A, but there is no mapping defined between the classes and the signatures of the operations in the classes do not match, the weaver terminates with an exception. The modeler is prompted to resolve the name conflict by either renaming the class in A or by defining an explicit mapping.
3. Weave: Merge model elements from B with model elements from A according to the instantiation mapping. Elements in B that are not mapped explicitly are simply copied into A. This yields the woven model A+B.
4. Update instantiations: Update the instantiations in B according to the rule described above and add them to A+B.

5 Conclusion

This paper presented TouchRAM, a multitouch tool for agile software design modeling aimed at developing scalable and reusable software design models. TouchRAM is available at `http://www.cs.mcgill.ca/~joerg/SEL/TouchRAM.html`. The tool highlights are: 1) a streamlined user interface that exploits mouse and touch-based input to enable intuitive and fast model editing, 2) a library of reusable design concern models, and 3) support for model interfaces and elaborate hierarchical model dependencies. With TouchRAM, a modeler can rapidly build complex software designs following either a top-down, bottom-up, or incremental design approach. The tool provides facilities to inspect different levels of abstraction of the design being modeled by navigating the model dependencies, to combine individual models in order to provide insight on how different models interact, as well as generate a complete woven model, if desired.

To the best of our knowledge, TouchRAM is currently the only AOM tool supporting aspect hierarchies. Unfortunately we were not able to verify that claim, since the other existing AOM tools are not readily available for the general public. These include: the Motorola WEAVR [7], which is a proprietary tool for modeling with the SDL notation, MATA [15], an AOM plugin for Rational Architect, and the UML/Theme tool [6].

We are currently working on providing export/import functionality to/from standard UML to integrate TouchRAM with other MDE tools used for software development. We are also planning on adding support for sequence diagrams and state diagrams to specify the behavior of software designs as described in [10]. Finally, we want to integrate some of the ideas of researchers from the HCI community that have worked on touch-based manipulations of diagrams in order to improve the TouchRAM interface. For instance, Frisch et al. [9] present a way for handling complex and large models by introducing off-screen visualization techniques in order to effectively navigate software models. The basic premise of their work is to represent model elements that are clipped from the current viewable area by proxies. Schmidt et al. [13] present several interesting multitouch interaction techniques designed for the exploration of node-link diagrams.

References

1. Kermeta. http://www.kermeta.org
2. Kompose. http://www.kermeta.org/mdk/kompose/
3. MT4j - Multitouch for Java. http://www.mt4j.org
4. Tangible User Interface Objects. http://www.tuio.org
5. Al Abed, W., Kienzle, J.: Information Hiding and Aspect-Oriented Modeling. In: 14th AOM Workshop, Denver, CO, USA, Oct. 4th, 2009. pp. 1–6 (October 2009)
6. Carton, A., Driver, C., Jackson, A., Clarke, S.: Model-driven theme/UML. Transactions on Aspect-Oriented Software Development (2008)
7. Cottenier, T., Berg, A.V.D., Elrad, T.: The motoroal weavr: Model weaving in a large industrial context. In: Industry Track of the 5th International Conference on Aspect-Oriented Software Development (AOSD'06). ACM, Bonn, Germany (2006)
8. Fleurey, F., Baudry, B., France, R., Ghosh, S.: A generic approach for automatic model composition. In: 11th AOM Workshop, Nashville, TN (2007)
9. Frisch, M., Dachselt, R.: Off-screen visualization techniques for class diagrams. In: 5th International Symposium on Software Visualization. pp. 163–172. ACM (2010)
10. Kienzle, J., Al Abed, W., Klein, J.: Aspect-Oriented Multi-View Modeling. In: AOSD 2009, March 1 - 6, 2009. pp. 87 – 98. ACM Press (March 2009)
11. Reddy, R., Ghosh, S., France, R.B., Straw, G., Bieman, J.M., Song, E., Georg, G.: Directives for composing aspect-oriented design class models. Transactions on Aspect-Oriented Software Development LNCS 3880, Springer-Verlag (2006)
12. Schmidt, D.C.: Model-driven engineering. IEEE Computer 39, 41–47 (2006)
13. Schmidt, S., Nacenta, M., Dachselt, R., Carpendale, S.: A set of multi-touch graph interaction techniques. In: ITS 2010. pp. 113–116. ACM (2010)
14. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional, 2nd edn. (2009)
15. Whittle, J., Jayaraman, P.: Mata: A tool for aspect-oriented modeling based on graph transformation. Lecture Notes In Computer Science 5002, 16–27 (2008)