TouchRAM: A Multitouch-Enabled Software Design Tool Supporting Concern-Oriented Reuse

Matthias Schöttle Omar Alam
Franz-Philippe Garcia Jörg Kienzle
School of Computer Science, McGill University,
Montreal, QC H3A 0E9, Canada
{Matthias.Schoettle | Omar.Alam |
Franz-Philippe.Garcia}@mail.mcgill.ca,
Joerg.Kienzle@mcgill.ca

Gunter Mussbacher
Department of Electrical and Computer Engineering,
McGill University, Montreal, QC H3A 0E9, Canada
Gunter.Mussbacher@mcgill.ca

Abstract

TouchRAM is a multitouch-enabled tool for agile software design modelling aimed at developing scalable and reusable software design models. This paper primarily focusses on the new features that were added to *TouchRAM* to provide initial support for concern-orientation, and then summarizes the new extensions to behavioural modelling and improved integration with Java. A video that demonstrates the use of *TouchRAM* can be found here:
`http://www.youtube.com/watch?v=l8LMqwwRPg4`

Categories and Subject Descriptors D.2.2 [*Software Engineering*]: Design Tools; D.2.10 [*Software Engineering*]: Design

Keywords concern-oriented software development, model composition, model interfaces, model hierarchies, model reuse, class diagrams, sequence diagrams, state diagrams

1. Introduction

TouchRAM is a multitouch-enabled tool for agile software design modelling 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 to enable the designer to rapidly apply reusable design concerns within the design model of the software under development. The user interface features of the tool are specifically designed for ease of use, reuse, and agility.

TouchRAM was introduced initially at SLE 2012 [1], and demonstrated subsequently at Modularity:aosd 2013 [9] and MODELS 2013 [11]. Since then, the main novelty is that *TouchRAM* has been extended to support concern-oriented software design as described in [3]. This paper briefly summarizes the main ideas behind concern-orientation in section 2, and then describes the main steps for providing basic support for concern-orientation within TouchRAM based on the Reusable Aspect Models (RAM) [10] approach in section 3. Section 4 highlights other new features that were added to *TouchRAM* in the last 6 months: message view editing and state view weaving as well as better integration with Java. Finally, the last section draws some conclusions and presents directions of future work.

MODULARITY '14, April 22 - 26 2014, Lugano, Switzerland.

ACM 978-1-4503-2773-2/14/04.
http://dx.doi.org/10.1145/2584469.2584475

2. Concern-Orientation

Concern-Driven Development (CDD) refers to software development approaches that combine the ideas of model-driven engineering (MDE), aspect-orientation, and software product lines with a strong emphasis on reuse. In contrast to classic MDE's concentration on models, the main element of focus in CDD is the *concern*. A concern is any domain of interest to a software developer[1]. A concern has a root phase, where the concern manifests itself for the first time, and encapsulates a *set of models*. These models describe the concern's properties with most appropriate modelling formalisms for all those phases of software development and those levels of abstraction required to sufficiently understand the concern. Some concerns appear in early phases of software development, e.g., broadly scoped system properties that can have functional, non-functional, or even intentional characteristics. In later phases, solution-specific concerns appear, e.g., specific communication protocols, concrete authentication algorithms, and design patterns. Each concern defines *model transformations* that link the models established for the concern across different levels of abstraction. Finally, a concern also encapsulates all relevant variations/choices that are available to software engineers at a given phase, together with guidance on how to choose among those variations by specifying the impact of each choice on softgoals and non-functional requirements.

A concern provides *three interfaces* [3]:

- The *Variation Interface* describes the available variations of the concern and the impact of different variants on high-level goals, qualities, and non-functional requirements. The variations are typically represented with a *feature model* [8] that specifies the individual features that a concern offers, as well as their dependencies (optional, alternative, requires, excludes). The impact of choosing a feature can be specified with goal models (e.g., i* [12], KAOS [5], GRL which is part of the User Requirements Notation (URN) standard [7], and the NFR framework [4]).
- The *Customization Interface* describes how a chosen variant can be adapted to the needs of a specific application. Each variant of a concern is described as generally as possible to increase

[1] This is different from aspect-oriented software development, where the word *aspect* is typically used to designate a crosscutting concern.

reusability. Therefore, some elements in the concern are only *partially* specified and need to be related or complemented with concrete modelling elements of the application that intends to reuse the concern. The customization interface is hence used when a specific variant of a reusable concern is *composed* with the application.
- The *Usage Interface* describes how the application can finally access the structure and behaviour provided by the concern. For example, the usage interface of the design model of a concern is typically comprised of all *public* classes and methods made available by the concern.

Consequently, to reuse a concern, a software engineer must 1) select the feature(s) with the best impact on relevant non-functional properties from the variation interface based on provided impact analysis, then 2) adapt the generated detailed models to the application context by mapping customization interface elements to application-specific model elements, to finally 3) use the behaviour provided by the selected concern features through the usage interface.

2.1 Example Concern: Observer

The *Observer Design Pattern* [6] is a good example of a design concern. There exist many different designs of the observer pattern in the literature. For instance, the notification message sent to the observers when the state of the subject changes can include the modified state (*Push*), or no data at all, and hence it is the responsibility of the observer to query the subject to get the changes (*Pull*). *Push* reduces the number of messages exchanged, whereas *Pull* can reduce the amount of data that is transferred. Also, in a single threaded design, in the case where there are a significant number of observers or when update operations require lengthy computations, state updates on subjects are slow. In that case, a multi-threaded implementation that executes notifications concurrently is a good alternative design that increases the speed at which a change is executed. Finally, there is an extended design of the observer pattern called *model-view-controller* (MVC) that defines additional *Controller* objects that react to events and then request state changes on the subject (here called *Model*), which in turn notifies the observer of the change (here called *View*). MVC has again two possible design strategies – *Active* and *Passive* – which differ in the way the control flow passes through the related controller, model, and view objects. Fig. 1 shows the variation interface of the *Observer* concern.

3. Concern Support in TouchRAM

TouchRAM is based on Reusable Aspect Models (RAM) [10], an aspect-oriented multi-view modelling approach for software design that integrates class diagram, sequence diagram, and state diagrams. A RAM model has a *customization interface* and a *usage interface* [2], but no support for expressing variability. As a result, a RAM model can describe the structure and the behaviour of *one specific* design solution, but not of an entire design concern that offers different features or design variants to the modeller. In other words, RAM was usable as is to describe the design of one feature, but the concept of concern that groups related features together was missing.

In order to add support for concerns into TouchRAM we therefore had to: a) extend the RAM metamodel to support concerns, features, and concern reuse; b) allow the modeller to create concerns and define features, as well as specify what RAM models describe each features' detailed design; c) update the existing RAM model library to concerns, and d) allow the modeller to reuse concerns, i.e., d1) to select the desired features of a concern, d2) to generate the detailed design of the chosen selection by composing all RAM models associated with the selected features together, d3) to use the customization interface to adapt the generic model elements to the specific context in which the concern is reused, and d4) to allow the application design to trigger behaviour provided by the concern. These extensions are described in more detail in the following subsections.

3.1 a) Concern-ifying the Metamodel

In the new version of *TouchRAM*, concerns have replaced aspects as the main unit of reuse, and the *TouchRAM* metamodel had to be updated in consequence. A specific design solution is still modelled with a RAM model, but all solutions related to a design concern are now grouped within a concern. A concern specifies a variation interface, which consists of a set of features. Each feature is realized by one or several RAM models.

Current Limitations: So far, the metamodel does not support the specification of impacts in the variation interface of a concern. Also, it is currently not possible to model feature dependencies, e.g., *requires* and *excludes*.

3.2 b) Creating Concerns in TouchRAM

Building a design concern is a non-trivial, time consuming task. It requires a deep understanding of the nature of the design concern to be able to identify the different features, to model the common properties and differences of the concrete solutions, and to express the impact of the different variants on high-level goals. This can only be done by a *domain expert*, i.e., someone experienced who fully understands the nature of the concern and the tradeoffs involved in the different available options.

TouchRAM now allows such a domain expert to create a new concern, name it, and then create a list of features that the concern offers. Each feature can be associated with one or several RAM models that describe its detailed design.

Current Limitations: Due to the limitations of the current metamodel, it is not possible in *TouchRAM* to specify the impacts of the features of a concern. Also, currently the features of a concern are shown graphically as a simple list, as opposed to a nice feature model representation with mandatory, optional, and alternative constraints among features.

3.3 Reusable Design Concern Library

Class libraries offer programmers thousands of classes that provide solutions for common implementation concerns, e.g., 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 behaviour 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 modelling. Its purpose is to increase modelling productivity by providing models for common design concerns that a modeller can use within an application model with minimal effort when appropriate.

The RDCML 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 as described in section 4.2.

The previous version of TouchRAM shipped with individual RAM models encapsulating the design of a workflow execution engine, network communication, and several design patterns. We restructured the RDCML and reorganized it by concerns: *a workflow concern* with features such as parallel and conditional execution, synchronization and timed execution, the *observer concern* described above, and an *association concern* that provides differ-

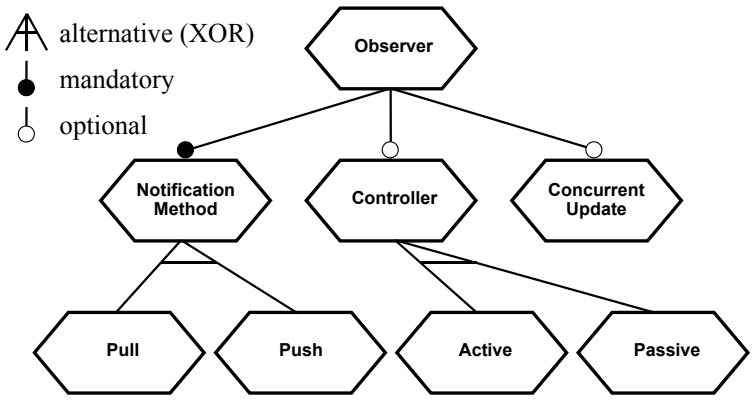

Figure 1. Features of the *Observer Concern*

ent ways of establishing one to many associations between design classes.

3.4 Reusing Design Concerns

The process of reusing a design concern is straightforward:

1. Use the variation interface of the concern to select the most appropriate feature, i.e., the feature that provides the desired functionality and maximizes positive impact on relevant non-functional application properties. This generates the detailed design for the selected feature(s) of the concern.
2. Use the customization interface of the generated design to adapt the generic design elements to the application-specific context. This generates the application-specific usage interface for the selected feature(s) of the concern.
3. Use the selected concern feature(s) within the application design according to the usage interface.

For instance, when reusing the *Observer* concern, the modeller looks at the available features exposed in the variation interface (see Fig. 1) and their impact on functionality, non-functional and high-level goals. She then determines that she wants to use pull notification (feature *pull*) with a passive controller (features *controller* and *passive*).

Currently, in TouchRAM, when a modeller wants to reuse a concern she is prompted to open the corresponding concern file, and is then presented with the list of features that the concern offers. Once the modeller decides on a variation, the *TouchRAM* weaver assembles all RAM models that realize the selected features and combines them according to the instantiation directives specified within the models. In our example, four RAM design models are combined, i.e., a) the one containing the common design properties among all observer variants, such as the `|Subject` and `|Observer` classes, b) the one with the `|getData` and `|update` operation needed for the *pull* feature, c) the one adding the `|Controller` class with its operations and associations and also renaming `|Subject` to `|Model` and `|Observer` to `|View`, as well as d) the one with the passive controller operations. The woven design class model is shown in Fig. 2. The customization interface is constituted of all classes and operations that have a "|" prefix. The usage interface is constituted of all the public operations (designated with a "+").

The last thing that is left to do is to customize the generated design to the application. For this, TouchRAM provides a special *mapping view* that simultaneously visualizes the application model and the generated concern variation. With gestures, the modeller can then, for example, specify the customization directives:

```
|Model → Stock; |modify → setNewPrice,
|getChanges → getPrice, |create → createStock,
|View → StockWindow, |updateView → refreshWindow,
|Controller → StockWindowController,
|handleEvent → processClick;
```

would allow a `StockWindow` instance to observe `Stock` instances and a `StockWindowController` instance to change the state of `Stock` instances.

4. Other Extensions to TouchRAM

TouchRAM has received several other significant extensions to existing parts, which are described briefly in the following subsections.

4.1 Extended Support for Behavioural Modelling

Previously *message views* (sequence diagrams) could only be visualized and woven in *TouchRAM*. The major extension is that *message views* can now also be created and edited. *TouchRAM* assists the user in the creation process. For example, when specifying a message call, only valid choices of operations (i.e., those defined in the *structural view*) that can be called are presented. Furthermore, the layout is done almost completely automatic. The user can only move lifelines and the messages are repositioned accordingly.

The second extension is weaving of state views (state diagrams). For every class there exists one state view, which in turn defines one or several state machines. When RAM models are woven together, the state views of the classes that are merged are composed using the *Communicating Sequential Processes* (CSP) parallel composition operator. A user can selectively decide to compose individual state machines, or weave them all together.

4.2 Extended Integration with Java

Enumeration types can now be defined in *TouchRAM*. For convenience, instead of having to add each literal separately, the user may provide the literals in one line by separating them with commas.

When designing software at a low-level, implementation details are important, and as such, the user might want to reuse existing classes provided by a programming language or a particular framework. We call these classes *implementation classes*. Currently, we support the Java language. The user informs *TouchRAM* to create an

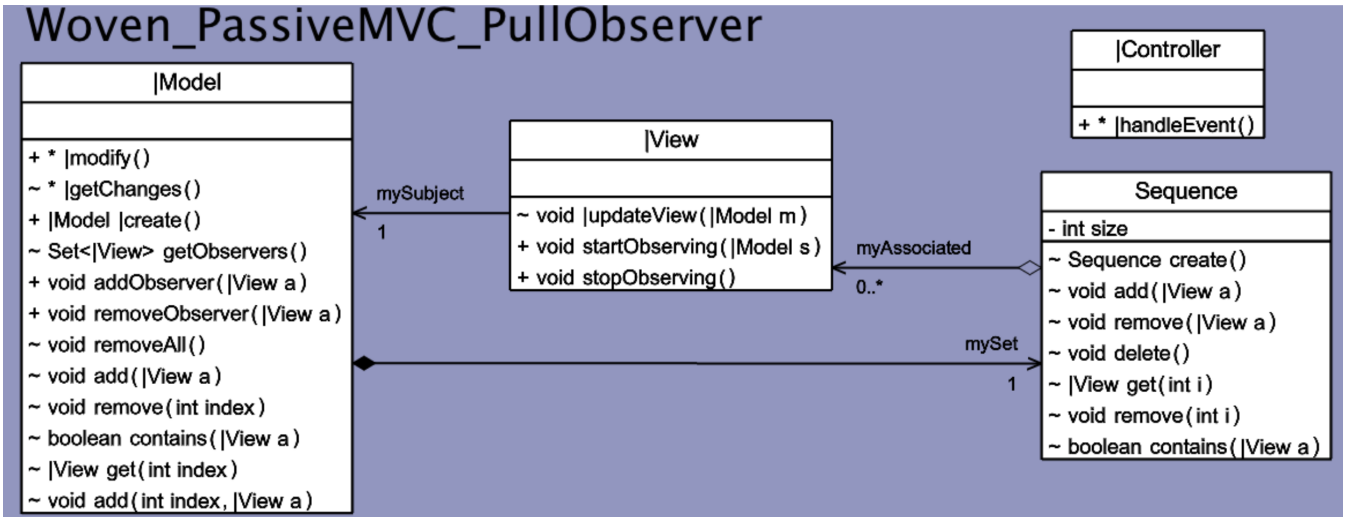

Figure 2. Generated Detailed Design for Observer<Pull,Controller,Passive> Selection

implementation class using a designated gesture. At the beginning, only classes from the Java library are available. In case the user wishes to import a class from a framework, the user may provide the JAR file. The user is presented with a choice once at least three characters of the desired class name are entered into the search field to avoid displaying too many options.

After selecting a class, it is visualized in the structural view. However, it is initially empty, i.e., it does not contain any operations. This is done because, in general, implementation classes have a large number of methods. We therefore decided to only show methods that the user actually wants to use. Therefore, when a particular method is needed (i.e., because the user wants to invoke it in a message view), the user can add it to the implementation class by selecting it from the list of methods that *TouchRAM* obtains using the Java reflection API. In case a method is selected with a return or parameter type that refers to other implementation classes that so far have not yet been imported, *TouchRAM* automatically adds them to the model.

5. Conclusion

This demo paper summarizes how *TouchRAM*, a multitouch-enabled tool for agile software design modelling, was extended in the last 6 months with initial support for concern-orientation, message view editing, state diagram weaving, and enhanced support for Java implementation classes. The current version of TouchRAM and the reusable design concern model library can be downloaded from `http://www.cs.mcgill.ca/~joerg/SEL/TouchRAM.html`. It runs on Windows, MacOS and Linux-based systems. To use the multitouch features of *TouchRAM*, a TUIO supported multitouch input device must be connected.

In future work, we are planning to augment the user interface capabilities of the tool so that it is possible to show the variation interface of a concern graphically with feature diagrams. Gestures should allow the modeller to assess the impact of a feature on high level goals directly within the *TouchRAM* tool. Furthermore, the creator of a concern should be able to specify features and impacts directly within *TouchRAM* and not as currently required in an external tool. Finally, with support for implementation classes now in place, we are planning to complete the code generation capabilities of *TouchRAM*.

6. Acknowledgements

This work was partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) as part of the Digital Surface Software Application Network (SurfNet).

References

[1] Al Abed, W., Bonnet, V., Schöttle, M., Alam, O., and Kienzle, J. TouchRAM: A multitouch- enabled tool for aspect-oriented software design. In *SLE 2012* (2012), LNCS 7745, Springer, pp. 275 – 285.

[2] Al Abed, W., and Kienzle, J. Information Hiding and Aspect-Oriented Modeling. In *14th Aspect-Oriented Modeling Workshop* (October 2009), pp. 1–6.

[3] Alam, O., Kienzle, J., and Mussbacher, G. Concern-oriented software design. In *MODELS*, LNCS 8107. Springer, 2013, pp. 604–621.

[4] Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. *Non-Functional Requirements in Software Engineering*. Springer, 2000.

[5] Dardenne, A., van Lamsweerde, A., and Fickas, S. Goal-directed requirements acquisition. *Science of Computer Programming 20* (1993), 3–50.

[6] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns*. Addison Wesley, Reading, MA, USA, 1995.

[7] International Telecommunication Union (ITU-T). Recommendation Z.151 (10/12): User Requirements Notation (URN) - Language Definition, approved October 2012.

[8] Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, S. Feature-oriented domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-21, SEI/CMU, 1990.

[9] Kienzle, J. Reusing software design models with TouchRAM. In *AOSD '13 Companion* (2013), ACM, pp. 23–26.

[10] Kienzle, J., Al Abed, W., and Klein, J. Aspect-Oriented Multi-View Modeling. In *AOSD 2009* (March 2009), ACM Press, pp. 87 – 98.

[11] Schöttle, M., Alam, O., Ayed, A., and Kienzle, J. Concern-oriented software design with touchram. In *MODELS-JP* (2013), no. 1115, CEUR, pp. 51 – 55.

[12] Yu, E. *Modelling strategic relationships for process reengineering*. PhD thesis, Department of Computer Science, University of Toronto, 1995.