Feature Modelling and Traceability for Concern-Driven Software Development with TouchCORE

Matthias Schöttle Nishanth Thimmegowda
Omar Alam Jörg Kienzle

School of Computer Science, McGill University,
Montreal, QC H3A 0E9, Canada
{Matthias.Schoettle, Nishanth.Thimmegowda,
Omar.Alam}@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

This demonstration paper presents TouchCORE, a multi-touch enabled software design modelling tool aimed at developing scalable and reusable software design models following the concern-driven software development paradigm. After a quick review of concern-orientation, this paper primarily focusses on the new features that were added to TouchCORE since the last demonstration at Modularity 2014 (were the tool was still called TouchRAM). TouchCORE now provides full support for concern-orientation. This includes support for feature model editing and different modes for feature model and impact model visualization and assessment to best assist the concern designers as well as the concern users. To help the modeller understand the interactions between concerns, TouchCORE now also collects tracing information when concerns are reused and stores that information with the woven models. This makes it possible to visualize from which concern(s) a model element in the woven model has originated.

Categories and Subject Descriptors D.2.10 [*Software Engineering*]: Design; I.6.5 [*Simulation and Modeling*]: Model Development

Keywords concern-driven software development, reuse, feature models, impact models, traceability.

1. Introduction

This demonstration paper presents TouchCORE, a multi-touch enabled software design modelling tool aimed at developing scalable and reusable software design models following the concern-driven software development paradigm. The predecessor of TouchCORE called TouchRAM was introduced initially at SLE 2012 [1], and later demonstrated at Modularity:aosd 2013 [2] and 2014 [3] and MODELS 2013 [4]. The most recent demonstration held at MODELS 2014 showcased how jUCMNav (a requirements modelling tool) and TouchRAM have been integrated to provide initial support for concern-driven software development based on a common metamodel [5].

As of early 2015, TouchCORE now provides full support for concern-driven software development as outlined in this paper. Section 2 briefly recalls the main concepts of concern-driven software development. Section 3 describes TouchCORE's support for feature model and impact model visualization and assessment targeted at concern designers as well as concern users. Section 4 explains how traceability support was added to the CORE (Concern-Oriented REuse) metamodel, which defines the key concepts of concern-driven software development, to allow visualization of the crosscutting nature of concerns in woven models. The last section draws our conclusions.

MODULARITY Companion'15, March 16–19, 2015, Fort Collins, CO, USA.

http://dx.doi.org/10.1145/

2. Concern-Driven Software Development

In contrast to the focus of classic Model-Driven Engineering (MDE) on models, the main unit of abstraction, construction, and reasoning in Concern-Driven Software Development (CDD) is the *concern* [6]. CDD seeks to address the challenge of how to enable broad-scale, model-based reuse. A concern is a unit of reuse that groups together software artifacts (models and code, henceforth called simply models) describing properties and behaviour related to any domain of interest to a software engineer at different levels of abstraction.

A concern provides a three-part interface. The *variation interface* describes required design decisions and their impact on high-level system qualities, both explicitly expressed using feature models and goal models in the concern specification. The goal models used in CDD are called impact models, because they describe the impact of variable features on high-level system qualities. The *customization interface* allows the chosen variation to be adapted to a specific reuse context, while the *usage interface* defines how the functionality encapsulated by a concern may eventually be used.

Building a concern is a non-trivial, time consuming task, typically done by or in consultation with a domain expert (subsequently called the *concern designer*). On the other hand, reusing an existing concern is extremely simple, and essentially involves 3 steps for the *concern user*: (1) Selecting the feature(s) of the concern with the best impact on relevant goals and system qualities from the variation interface of the concern. (2) Adapting the general models of features of the concern that were selected to the specific application context based on the customization interface. (3) Using the functionality provided by the selected concern features as defined in the usage interface within the application.

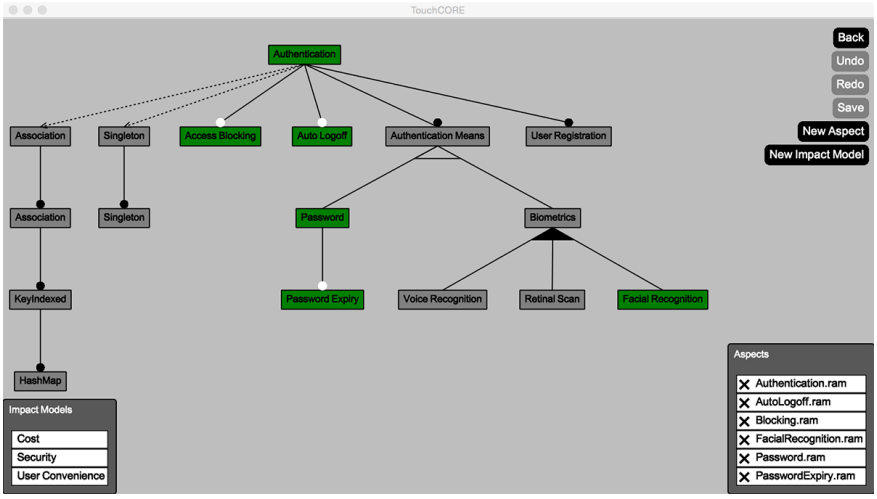

Figure 1. Visualization of Feature Model for Concern Designer

In general, MDE approaches rely heavily on tool support. Tool support is even more important in the context of CDD, in particular for the concern user:

- When selecting the set of features of a concern that best meets the requirements of the application under development, a concern user needs to be able to perform trade-off analysis between different variations/implementations of the needed functionality. To do that efficiently, a tool is needed that performs real-time impact analysis of feature selections. TouchCORE now provides this functionality as described in Section 3.
- Once a selection is made, a tool is needed that composes the models that realize the selected features to yield tailored models of the concern corresponding to the desired configuration.
- When adapting the generated concern models to the application context, the concern user must map customization interface elements from the concern to application-specific model elements in the application. Tool support is helpful to ensure that the mapping is specified correctly.
- Once the concern model is customized, a tool can help to ensure that the functionality provided by the concern is correctly used.

3. Support for Feature Models and Impact Models

In the context of concern-driven development, *feature models* capture the relationships and dependencies that exist between distinctive user-visible aspects and characteristics of the software that a concern modularizes and encapsulates. *Impact models* describe the impact that choosing a particular feature has on non-functional properties and qualities by means of goal models.

3.1 Visualization for the Concern Designer

The *concern designer*, i.e., the developer that creates the models and code encapsulated within a concern, uses the feature model to organize the features in a hierarchical "table of contents" that facilitates reasoning about structural and behavioural dependencies of the artifacts that realize the features encapsulated within the concern. Therefore, in the concern-design mode, the TouchCORE tool always displays the entire feature model of the current concern to the concern designer (see Fig. 1).

In order to be able to do his job well, the concern designer needs to not only be aware of all relevant information regarding the organization of the current concern, but also which other concerns this concern (or one of its features) reuses. To document the reuse clearly and to communicate to the designer that he can depend on functionality provided by the selected features of the reused concerns, if desired, TouchCORE *visualizes the selected features of a reused concern as mandatory descendants of the feature of the current concern that made the reuse*. In CDD, it is possible to delay the selection of variable features from a reused concern. In this case, the concern designer of the current concern C does *not* make a decision regarding variable features of the reused concern R, but reexposes them to the concern user who will reuse the current concern C in his new concern N and decide which reexposed features to choose (or to reexpose them again). The features of the reused concern that are reexposed by the current concern are not visualized in the view of the concern designer (but they are shown in the view of the concern user). This facilitates the task of the designer of the current concern, since he is allowed to use the structure and behaviour provided by selected features, but not by reexposed ones.

TouchCORE also allows the concern designer to associate impact models with the concern. Each impact model specifies quantitively how each feature that the concern encapsulates affects a high-level goal or non-functional quality.

3.2 Visualization for the Concern User

The *concern user*, i.e., the developer that reuses an existing concern within the software under development, consults the feature model of a concern to discover the available functional variants and design choices that the concern offers, and the impact each alternative has on non-functional software properties. To make a choice, he needs to maximally focus on deciding which variant offered by the concern is best. Therefore, in the concern-reuse mode, the TouchCORE tool only displays the optional choices (features with optional, OR, and XOR dependencies) including reexposed features of the concern that is being reused, omitting mandatory features or features that are required/excluded by the current selection (see Fig. 2).

Whenever the current selection changes, TouchCORE automatically evaluates the impact models of the concern and presents the resulting satisfaction values of each high-level goal or quality to the concern user to enable trade-off analysis. The interested reader is

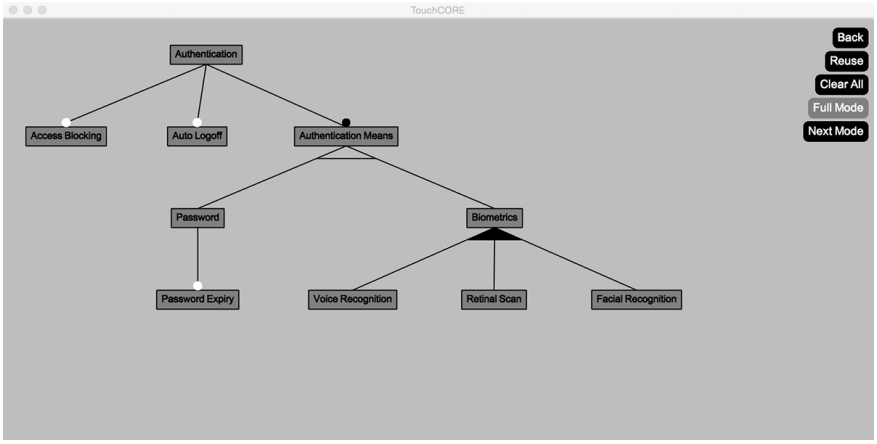

Figure 2. Visualization of Feature Model for Concern User

referred to [7] for a detailed description of the feature model visualization algorithms.

4. Traceability Support

Complex applications consist of many intertwined, interacting concerns, and CDD advocates to develop an application by reusing as many already existing concerns as possible. The same principle applies to the development of a concern itself. Typically, a requirements concern, e.g., security, needs to comprise not only models that specify different ways of achieving security (authentication, role-based access control, encryption, etc.), but also different ways of realizing them (password-based authentication vs. biometrics, etc.). For a given realization, there are different possible implementation architectures (centralized password server vs. local, distributed databases, etc.). It comes with no surprise that low-level design solutions, such as various design patterns, transaction controls, or resource allocation are quite general solutions that can be reused in many designs.

To fully reap the benefits of reuse, CCD supports the creation of *concern hierarchies*. To increase scalability and avoid duplication of effort, a high-level concern (or to be more precise, a feature of a high-level concern) can reuse the functionality (structure/behaviour/properties) of a lower-level concern when appropriate. Similarly, a more domain-specific or solution-specific concern can reuse other more general concerns.

Concern hierarchies allow the modeller to modularize the application into different layers of abstraction. In order to reduce complexity, concerns allow for separate reasoning, and hide the complexity of the lower levels from the upper levels. Concerns can be focussed on in isolation, and mappings between model elements in different concerns are established to connect their structure and behaviour when needed.

However, separation of concerns also makes it harder to understand the detailed interaction between different concerns. This is why traceability support is important in tools that support CCD, and was recently integrated into TouchCORE.

4.1 Traceability Support in the Metamodel

Fig. 3 shows how the CORE metamodel was extended to support tracing. Every *COREModel* can now contain many *COREWovenModels*. An instance of *COREWovenModel* has a name (inherited from *CORENamedElement*), a reference that references the *COREModel* that has been woven into the current model, and a set of *CORETraceableElements*, which can, for example, be attributes, operations or classes contained in the structural view.

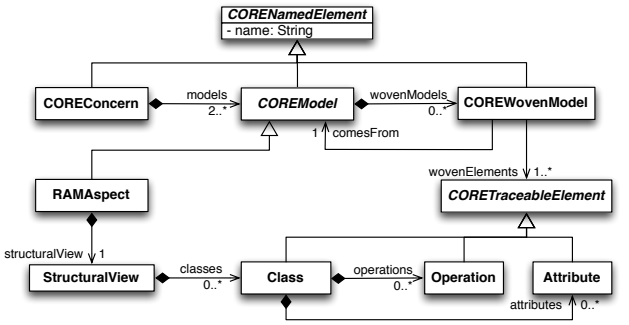

Figure 3. Tracing Integration into the (simplified) CORE Metamodel

4.2 Traceability Support in the Weaver

In TouchCORE, weaving is performed in pairs. Whenever the weaver is asked to weave model *A* into model *B*, it duplicates *B* together with all the model elements it contains and names it *Woven_B*. Then, all the model elements from *A* are copied into this new model. In the post-processing phase, the weaver then decides if certain structural model elements have to be merged. This could be the case, if the modeller provided customization mappings between A and B, and also for model elements with matching signatures in the case where *A* and *B* realize features of the same concern. The interested reader can find a detailed description of the structural weaving algorithm in [1]. Behavioural weaving is described in [8].

To support tracing, the post-processing phase of the weaver was extended as follows. Once the weaving is completed, a new instance of *COREWovenModel* is created, its name initialized to the model that was woven (i.e., *A* in our example) to obtain the name in case the original model cannot be accessed, and the *comesFrom* reference is set to point to *A*. Then, the set *wovenElements* is set to contain all the elements that were either copied from *A* directly, or that were merged with elements from *A*.

4.3 Traceability Visualization

Whenever a model contains information about other models that were woven into it, a tracing view is shown as seen in Fig. 4 on the bottom left. The view lists all *COREWovenModels* by name. To see which elements were woven from a certain model, the user can select one or more entries and the tool highlights the corresponding woven elements in different colours. This allows

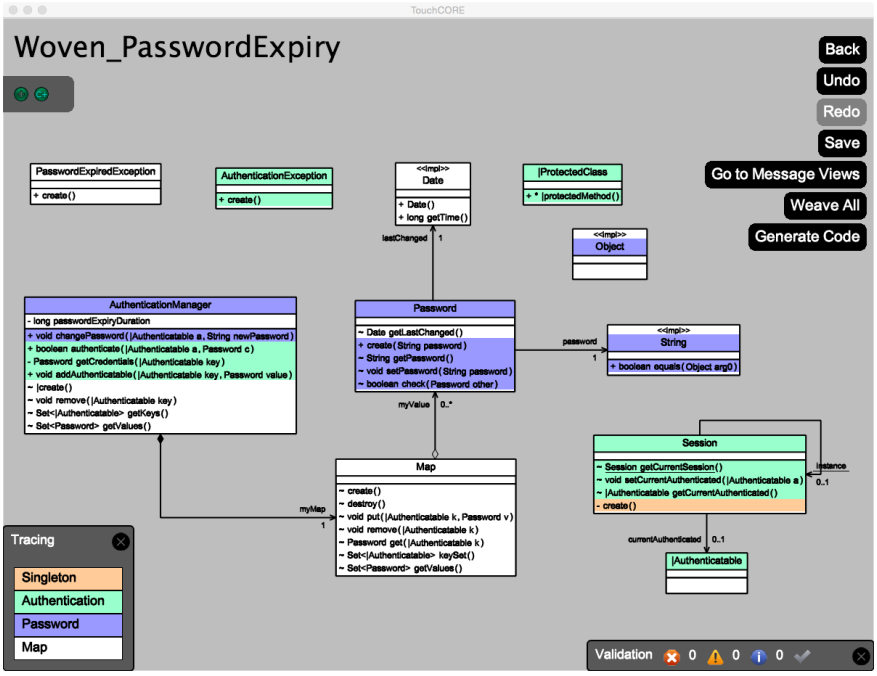

Figure 4. Visualization of Tracing Information

the user to understand where certain elements came from and, if necessary, address issues.

5. Conclusion

This demonstration paper presented TouchCORE, a multi-touch enabled software design modelling tool aimed at developing scalable and reusable software design models following the concern-driven software development paradigm. Its predecessor, TouchRAM, already supported the specification of detailed design models using class, sequence and state diagrams. Furthermore, TouchRAM enabled the definition of reusable design models using aspect-oriented technology such as class merging, sequence diagram weaving, and state diagram composition. TouchCORE builds on top of this technology, and now provides full support for concern-driven software development for both concern designers and concern users.

The concern designer can define a concern that encapsulates all relevant variations/design choices, and express the variability with a feature model. Guidance on how to choose among those variations can now be specified with impact models that describe how each feature affects high-level goals and non-functional requirements. TouchCORE assists the model designer by displaying the entire feature model of the concern being designed, and additionally visualizes selected features of reused concerns as mandatory child features. On the other hand, the concern user is mainly interested in discovering the available variants and design choices that the concern offers and their impacts. Therefore, TouchCORE presents a simplified feature model that only displays the available choices including reexposed features to the concern user when a reuse is initiated. In this case, the full impact model is hidden, and only the evaluated satisfaction values for each high-level goal are shown when the concern user selects the features he wishes to reuse.

Finally, TouchCORE now also gathers tracing information when generating complex woven models combining multiple concerns and features. The tracing information is useful, e.g., for debugging reasons, since it allows the modeller to visualize for each model element from which concerns/features it originated. Furthermore, the visualized tracing information provides valuable insight into the interactions between different concerns and their crosscutting nature.

References

[1] W. Al Abed, V. Bonnet, M. Schöttle, O. Alam, and J. Kienzle, "TouchRAM: A multitouch-enabled tool for aspect-oriented software design," in *SLE 2012*, no. 7745 in LNCS, pp. 275 – 285, Springer, October 2012.

[2] J. Kienzle, "Reusing Software Design Models with TouchRAM," in *Companion of Modularity:AOSD 2013*, pp. 23–26, ACM, March 2013.

[3] M. Schöttle, O. Alam, F.-P. Garcia, G. Mussbacher, and J. Kienzle, "TouchRAM: A Multitouch-enabled Software Design Tool Supporting Concern-oriented Reuse," in *Companion of Modularity:2014*, pp. 25–28, ACM, 2014.

[4] M. Schöttle, O. Alam, A. Ayed, and J. Kienzle, "Concern-Oriented Software Design with TouchRAM," in *Proceedings of the Demonstrations Track of the International Conference on Model Driven Engineering Languages and Systems (MODELS 2013)*, vol. 1115 of *CEUR Workshop Proceedings*, pp. 1 – 6, October 2013.

[5] N. Thimmegowda, O. Alam, M. Schöttle, W. Al Abed, T. Di'Meco, L. Martellotto, G. Mussbacher, and J. Kienzle, "Concern-Driven Software Development with jUCMNav and TouchRAM ," in *Proceedings of the Demonstrations Track of the International Conference on Model Driven Engineering Languages and Systems (MODELS 2014)*, vol. 1255 of *CEUR Workshop Proceedings*, pp. 1 – 6, 2014.

[6] O. Alam, J. Kienzle, and G. Mussbacher, "Concern-oriented software design," in *International Conference on Model-Driven Engineering Languages and Systems - MODELS 2013*, vol. 8107 of *LNCS*, pp. 604–621, Springer, 2013.

[7] N. Thimmegowda and J. Kienzle, "Visualization Algorithms for Feature Models in Concern-Driven Software Development," in *Short Paper accepted at Modularity 2015*, pp. 1 – 4, to be published.

[8] M. Schöttle, "Aspect-Oriented Behavior Modeling In Practice," M.Sc. Thesis, Department of Computer Science, Karlsruhe University of Applied Sciences, September 2012.