Support for Evaluation of Impact Models in Reuse Hierarchies with jUCMNav and TouchCORE

Romain Alexandre*, Cécile Camillieri*, Mustafa Berk Duran†,
Aldo Navea Pina†, Matthias Schöttle‡, Jörg Kienzle‡, and Gunter Mussbacher†
*Polytech Nice Sophia, Université de Nice, France
{romain.alexandre | cecile.camillieri}@polytech.unice.fr
† Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada
{berk.duran | aldo.naveapina}@mail.mcgill.ca, gunter.mussbacher@mcgill.ca
‡School of Computer Science, McGill University, Montreal, QC, Canada
Matthias.Schoettle@mail.mcgill.ca, Joerg.Kienzle@mcgill.ca

***Abstract*—In Concern-Orientation, software systems are built with the help of reusable artifacts called *concerns*, leading to *reuse hierarchies*, because higher-level concerns may reuse lower-level concerns. At each level in the reuse hierarchy, a concern uses goal modelling techniques to describe the impact of selected variations from the concern on system qualities such as performance, cost, and user convenience. To reason about trade-offs among system qualities in the whole system, the individual goal models from all levels in the reuse hierarchy have to be considered together. This requires the ability to select variations from different levels in the reuse hierarchy, to connect impacts from lower levels to those at higher levels, and eventually to propagate the evaluation of lower-level goal models to higher-level goal models based on the selection of variations. This tool demonstration reports on such an evaluation mechanism for two tools that provide integrated support for Concern-Orientation: the requirements engineering tool *jUCMNav* and the software design tool *TouchCORE*.**

I. Introduction

Model-Driven Engineering (MDE) advocates the use of different modelling formalisms during software development, so that the most appropriate modelling notation at the right level of abstraction can be chosen to describe and reason about the system under development. In *Concern-Orientation*, software systems are built with reusable artifacts called concerns. A *concern* is a new model-based unit of reuse encapsulating software artifacts pertaining to a domain of interest that span multiple development phases and levels of abstraction [1]. Hence, various modelling notations are used to describe a concern covering structural, behavioural, and other properties as required by the domain of interest. However, a concern always uses (i) feature models [2] to capture the reusable variants/solutions provided by the concern and (ii) impact models to capture the impact of a selection of features on system qualities.

This tool demonstration focuses on the impact models of concerns, which are based on goal modelling technology [3], [4], [5] and depend on the selection of features. A different feature selection may result in drastically different impacts on system qualities. Concern-Orientation aims to address a shortcoming of popular reusable artifacts, such as Java libraries or components, which excel in defining an API or required/provided interfaces that allow the reusable artifact to be used, but fail when it comes to describing which reusable artifact should be chosen over another reusable artifact. A developer typically has to discover the advantages and disadvantages of candidate reusable artifacts by going through documentation, blogs, and tutorials. In Concern-Orientation, this knowledge is captured in the impact model of a concern.

Concern-Orientation advocates the reuse of smaller, lower-level concerns to build larger, higher-level concerns leading to a *reuse hierarchy* consisting of concerns differing in size, abstraction level, and complexity. When a concern is reused, the impacts of selected features on system qualities must consequently be connected to the impacts of the reusing concern, so that eventually the whole system can be analyzed. The connected impacts must allow for the propagation of evaluation values from lower-level goal models to higher-level goal models based on the selection of variations from different levels in the reuse hierarchy. This tool demonstration reports on such an evaluation mechanism in the requirements engineering tool *jUCMNav* [6] and the software design tool *TouchCORE* [7]. Both tools provide integrated support for Concern-Orientation based on a common metamodel [8]. Out of the many tool demonstrations for jUCMNav, TouchCORE, and its predecessor *TouchRAM*, there are three demonstrations that are most relevant to this demonstration. At RE 2014, combined reasoning of goal and feature models was presented with jUCMNav [9]. At MODELS 2014, a demonstration showcased how jUCMNav and TouchRAM have been integrated to provide initial support for Concern-Orientation [8]. At Modularity 2015, the TouchCORE tool was demoed for the first time, focusing on feature modelling and traceability in the context of Concern-Orientation [10]. However, the evaluation of impact models across levels in reuse hierarchies, i.e., across concern boundaries, has not been demonstrated before.

Section II gives a brief overview of the evaluation of goal models and outlines which challenges need to be addressed for the evaluation of goal models in reuse hierarchies. Section III describes the required changes to the jUCMNav and TouchCORE tools in support of goal model evaluation in reuse hierarchies, while the last section concludes the paper and discusses future work.

II. Shortcomings of Propagation-Based Evaluation of Goal Models

Goal modelling notations such as the NFR Framework [3], the Goal-oriented Requirement Language (GRL), which is part of the User Requirements Notation [4], or i* [5] support reasoning about goals and system qualities through propagation-based evaluation mechanisms. Such evaluation mechanisms propagate user-defined initial satisfaction values from lower-level goal model elements—typically leaf nodes—to those of higher-level goal model elements. Both jUCMNav and TouchCORE use goal modelling technology based on GRL. In the context of Concern-Orientation, lower-level goal model elements are typically features of the concern (i.e., the different variations offered by the concern) and higher-level goal model elements describe system qualities. Furthermore, the initial satisfaction value of a feature is either the minimum possible (i.e., `0` in our case) if the feature is not selected and the maximum possible (i.e., `100` in our case) if the feature is selected.

There are three challenges that need to be addressed to enable propagation-based evaluation across concern boundaries in a reuse hierarchy.

a) Distributed Impact Models: Existing propagation-based evaluation mechanisms assume that a goal model is contained in one module and not distributed over the impact models of several reusable concerns. Using existing goal modelling concepts, one could theoretically create a single, monolithic impact model that contains the individual impact models from all levels in the reuse hierarchy. While this would defeat the purpose of having multiple, reusable modules in the first place, there is a second reason why this is problematic. This monolithic impact model would have to be able to dynamically discount impacts from lower-level concerns, if the feature that is reusing the lower-level concern is currently not selected. This is not easily possible with standard goal modelling concepts. We therefore introduced a new goal modelling element called the *Feature Impact Node* [11], which aggregates the impacts of reused concerns and the reusing feature, takes into account whether the reusing feature is selected or not to actually propagate upwards its satisfaction value, and still respects concern boundaries. An improved evaluation mechanism for Concern-Orientation has to properly take into account Feature Impact Nodes.

b) Distributed Specification of Initial Satisfaction Values: Fig. 1 combines three levels of concerns in a reuse hierarchy into one feature model. At the top, the features of a *Bank* concern are shown with white background. Directed dotted arrows in the feature model indicate the reuse of a concern. The *Authorization* concern is reused in total three times: by the *Internet Banking Interaction* feature, the *ATM Interaction* feature, and the *Vault* feature. In addition, the *Camera* and *Sensor* concerns are also reused by *the Vault* feature. The *Authorization*, *Camera*, and *Sensor* concerns are hence at the second level in the reuse hierarchy. The *Authorization* concern itself reuses the *Authentication* concern, i.e., *Authentication* is at the third level in the reuse hierarchy. Consequently based on this reuse hierarchy, *Authorization* first reuses *Authentication*, and in turn *Authorization* is then reused by the *Bank*.

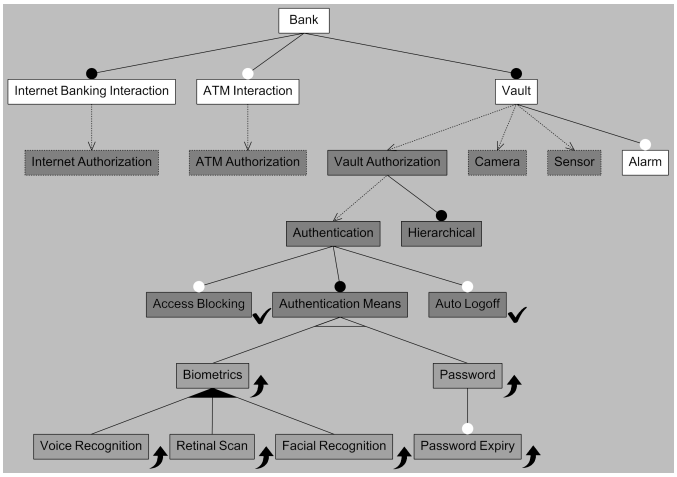

Figure 1. Combined Feature Model of Reuse Hierarchy with Three Levels of Concerns (Dynamically Visualized with TouchCORE)

At the time *Authorization* reuses *Authentication*, the developer of *Authorization* cannot fully select all necessary features in *Authentication*. For example, the *Password* feature and the *Facial Recognition* feature both work for what *Authorization* wants to accomplish, but their impacts on system qualities cause the first selection to be preferable if cost is an issue and the other selection to be preferable if tighter security is an issue. However, the developer of *Authorization* is not in a position to make this decision, because this depends on the context of whoever is going to reuse *Authorization* (i.e., only the developer of the *Bank* in our case can determine that). Therefore, the developer of *Authorization* will only make a partial selection of features in *Authentication* and leave some features open to be selected by the developer of the *Bank* (i.e., the developer of *Authorization* is delaying some decisions to whoever is going to reuse *Authorization*).

In Fig. 1, the checkmarks indicate that the developer of *Authorization* selected the *Access Blocking* and *Auto Logoff* features from the *Authentication* concern, because the developer deemed them to be absolutely necessary for the purposes of *Authorization*. This decision may be debatable, but serves to illustrate a point with the example. The up arrows, on the other hand, indicate those feature that can still be selected by the developer of the *Bank*. Remember that feature selections are inputs to the impact model, which contain features as leaf nodes. Consequently, the user-defined initial satisfaction values corresponding to the selection of features are distributed over several concerns. Existing propagation-based evaluation mechanisms assume that this is not the case. To address this issue, we introduced the ability to specify a partial set of initial satisfaction values [11]. An improved evaluation mechanism for Concern-Orientation has to combine these partial sets into one set for each impact model before performing the evaluation of the impact model in the reuse hierarchy.

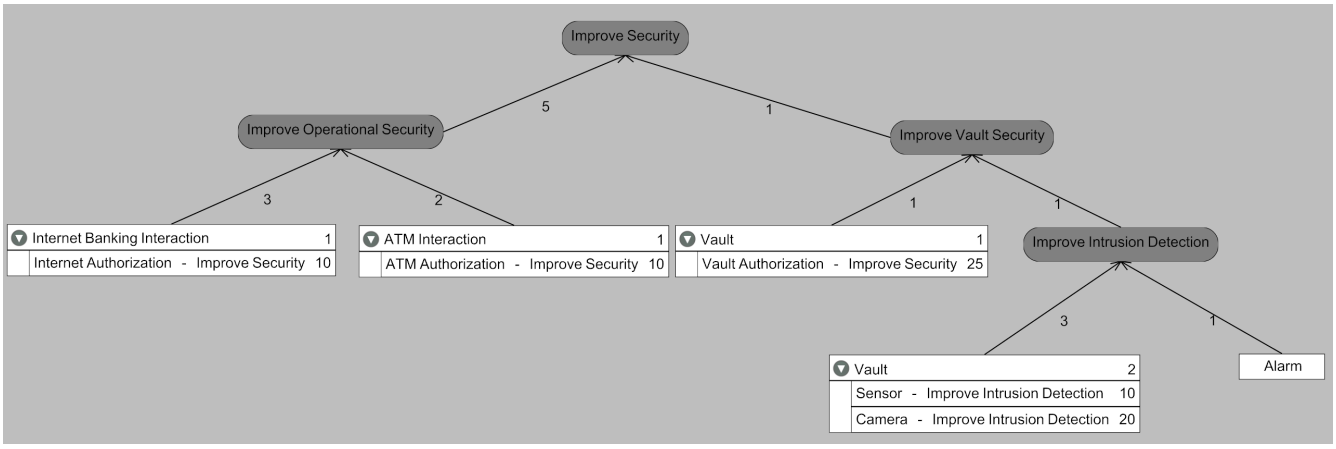

Figure 2. Reuse of Impact Model Elements in TouchCORE

c) Relative Contribution Values: The third issue is orthogonal to the first two already mentioned ones. It is related to the use of relative contribution values in impact models. In goal models, contribution values are defined for contribution links that connect goal model elements and propagate the satisfaction values of source elements to a parent element with a weighted sum approach.

For the NFR Framework, GRL, and i*, three types of contribution values are known: qualitative, quantitative, and real-life values. Qualitative values are too imprecise for use in Concern-Orientation, because they only differentiate features by limited shades of negative and positive contributions. Quantitative values are from a specific range (typically `[-100,100]`). In the context of Concern-Orientation, they are difficult to use, because one has to assume that many developers will build reusable concerns. If a developer of Concern *A* describes the impact on a system quality as `75` out of `[-100,100]` and a developer of Concern *B* describes the impact on the same system quality as `90` out of `[-100,100]`, then there has to be an implicit understanding what these numbers mean. If not, the two concerns cannot be reused together, because the inconsistency in the contribution values would render the impact model analysis useless. Given that impacts need to be specified to vague system qualities such as *user convenience*, *usability*, or *security*, it is unlikely that such an understanding exists. The third option (real-life values) addresses this issue, because actual measurements from real-life are taken to characterize the impact on a system quality. While this addresses the inconsistency issue, it is not necessarily easy or cost-effective to capture such measurements.

Therefore, we have devised a way to use relative contribution values [12]. This allows each developer to specify contribution values relative to all locally known impacts on a system quality. Furthermore, an improved evaluation mechanism for Concern-Orientation has to ensure that the satisfaction values, which are derived from the relative contribution values, are normalized to the `[0,100]` range, thus enabling the composition of impact models from different concerns. In addition, the normalization has to take constraints among impact model elements into account [12].

III. Evaluation in Reuse Hierarchies with jUCMNav and TouchCORE

jUCMNav is a requirements engineering tool that already offers significant support for the evaluation of impact models, but until now had little support for model reuse. TouchCORE, on the other hand, is a software design tool that is predominantly built in support of model-based reuse, and until now had no support for goal-oriented modelling. Consequently, different functionality had to be added to the tools to support distributed impact models, distributed specification of initial satisfaction values, and relative contribution values.

In the TouchCORE tool, support for impact modelling was implemented from scratch, taking best practices from GRL into account (e.g., a clear distinction between the impact model definition for a concern and views of those definitions, allowing the same impact model element to be shared among views focusing on one high-level system quality at a time). Fig. 2 depicts an impact model for improving the security of the *Bank* concern as defined in the TouchCORE tool. The contributions are defined with relative values and four new Feature Impact Nodes (FINs) are used in the impact model. Each FIN is related to exactly one feature of the *Bank* concern as shown in the top row of the FIN. Each additional row of a FIN represents a reused impact model element from another concern. The numbers in each row describe the relative impact of the reused impact model elements and the feature itself.

In jUCMNav, the concept of a *Reuse Link* was introduced to allow impact model elements from a reusable concern to be used in the impact model of the reusing concern. A Reuse Link is illustrated as a dashed arrow in Fig. 3, which shows the same impact model from Fig. 2 as it is visualized in the jUCMNav tool. Note how each FIN is not represented with a rectangle as in TouchCORE, but is visualized with goal model elements that are already supported by jUCMNav (the representation used by TouchCORE may also be implemented for jUCMNav in the future). For example, the two goals

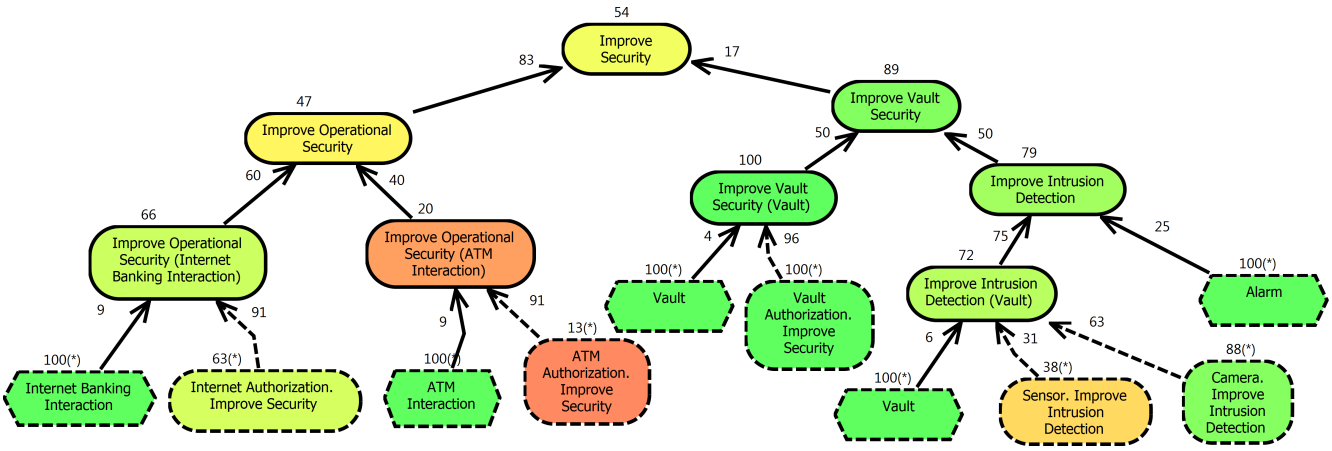

Figure 3. Evaluation of Impact Models with Reuse Hierarchy in jUCMNav

Improve Operational Security (Internet Banking Interaction) and *Internet Authorization.Improve Security* as well as the feature *Internet Banking Interaction* correspond to the leftmost rectangle in Fig. 2.

The evaluation of impact models in a reuse hierarchy uses an algorithm that starts at the topmost impact model and then recursively evaluates all impact models of reused elements before evaluating the impact model with the reused elements. Feature selections from various levels in the reuse hierarchy are combined as needed. Fig. 3 shows the result of an evaluation of impact models in a reuse hierarchy with the contribution values as well as satisfaction values after normalization (e.g., the `1` and `10` from the Internet Banking Interaction FIN in Fig. 2 are translated into a `9` and `91` on the `[0,100]` scale in Fig. 3).

For the normalization of satisfaction values calculated with relative contribution values, the improved evaluation mechanism needs to check whether any constraints among features are violated given a specific feature selection. This is accomplished with the help of the FAMILIAR tool [13][14], which uses a SAT solver to verify whether constraints in a feature model are violated.

IV. Conclusion

This tool demonstration showcases support for the evaluation of impact models of reusable concerns in a reuse hierarchy in two tools: *jUCMNav*—a requirements engineering tool with a long history in goal-based modelling, and *TouchCORE*—a software design tool with a long history in model-based reuse. In *Concern-Orientation*, the impact of a concern's features on system qualities is captured in a specialized goal model called impact model. With the new support, it is now possible to evaluate the combined impact of all concerns in a reuse hierarchy to enable system-wide trade-off analysis of candidate feature selections across concern boundaries. This is an improvement over existing propagation-based evaluation mechanisms for goal models, which typically assume a monolithic goal model with a centralized specification of initial satisfaction values. In addition, it is now possible to specify relative contribution values in impact models. In the future, we will consider extending impact models (i) with other goal modelling concepts such as actor (for the modelling of intentions of stakeholders) and indicator (for the modelling of real-life measurements) and (ii) with better support for taking contextual information from the reusing concern into account in the impact model of the reused concern.

References

[1] O. Alam, J. Kienzle, and G. Mussbacher, "Concern-Oriented Software Design," in *MODELS 2013*, pp. 604–621, Springer, 2013.

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

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

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

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

[6] "jUCMNav Tool." http://jucmnav.softwareengineering.ca/.

[7] "TouchCORE Tool." http://touchcore.cs.mcgill.ca/.

[8] 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, October 2014.

[9] Y. Liu, Y. Su, X. Yin, and G. Mussbacher, "Combined goal and feature model reasoning with the user requirements notation and jucmnav," in *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pp. 321–322, Aug 2014.

[10] M. Schöttle, N. Thimmegowda, O. Alam, J. Kienzle, and G. Mussbacher, "Feature modelling and traceability for concern-driven software development with TouchCORE," in *Companion Proceedings of MODULARITY 2015*, pp. 11–14, ACM, 2015.

[11] M. B. Duran, A. Navea Pina, and G. Mussbacher, "Evaluation of Reusable Concern-Oriented Goal Models," *Model-Driven Requirements Engineering (MoDRE)*, 2015.

[12] M. B. Duran, G. Mussbacher, N. Thimmegowda, and J. Kienzle, "On the Reuse of Goal Models," *SDL Forum'15*, October 2015.

[13] M. Acher, P. Collet, P. Lahire, and R. B. France, "Familiar: A domain-specific language for large scale management of feature models," *Science of Computer Programming (SCP)*, vol. 78, no. 6, pp. 657–681, 2013.

[14] "FAMILIAR Tool." http://familiar-project.github.io/.