Starting Model Development in Distributed Teams with Incremental Model Splitting

Daniel Strüber, Gabriele Taentzer
Philipps-Universität Marburg
{strueber, taentzer}@mathematik.uni-marburg.de

Abstract: The rising impact of software development in globally distributed teams strengthens the need for strategies that establish a clear separation of concerns in software models. Large, weakly modularized models are hard to comprehend and to analyse. A further maintainance obstacle is introduced by conflicting changes of the model. In our recent work, we propose a structured process for distributed modeling based on a variety of distributed modeling activities. To allow modularity and to facilitate developer independence, we focus on the problem of splitting a large monolithic model into sub-models. The modeler is assisted in incrementally discovering the set of desired sub-models. Our approach is supported by an automated tool that performs model splitting using information retrieval and model crawling techniques. We demonstrate the effectiveness of our approach on a set of real-life case studies, involving UML class models and meta-models being based on the Eclipse Modeling Framework.

1 Introduction

Model-based engineering – the use of models as the core artifacts of the development process – has become an industrially accepted best practice in many application domains. Together with the increased popularity of modeling, models of practical use grow in size and complexity to the point where large monolithic models are difficult to comprehend and maintain. There is a need to *split* such large models into a set of dependent modules (a.k.a. *sub-models*) and to modify these modules in a systematic way, increasing the overall comprehensibility and allowing multiple distributed teams to focus on each sub-model separately. The realization of a distributed model-driven software development approach is the aim of a recently started research project funded by the German Research Council.

2 Towards a Distributed Modeling Process for Composite Models

In [STJS13], we propose a structured process for distributed modeling. In order to untangle intransparent dependencies between sub-models, we base the process on a modularization technique called *composite models*. In composite models, the user defines interrelations between sub-models using explicit export and import interfaces. The explicit dependency management brought by composite models allows us to reason about *asynchronous* and *synchronous* editing steps in a distributed modeling team. We formally introduce the splitting, editing and merging of composite meta-models and model instances. While we show the soundness of splitting a model instance according to the split of its meta-model, we leave the splitting of the meta-model as a manual step to the meta-model developer .

3 Incremental Model Splitting

In [SRTC14], we propose a heuristic approach that allows splitting meta-models and model instances into sub-models. The target sub-models are specified by the user upfront using natural-language descriptions. In the core of the new approach, outlined in Fig. 1, is an automated technique that applies *information retrieval* and a topological analysis called *model crawling*. The user sets a *completeness condition* specifying whether the input set of sub-model descriptions is complete: For a complete set, the resulting *splitting suggestion* assigns each model element to exactly one sub-model. For an incomplete set, some elements are not assigned. The user can inspect the unassigned elements to discover additional sub-models, incrementally creating a complete split.

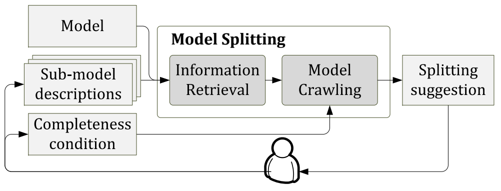

Figure 1: Model splitting using *information retrieval* and *model crawling* techniques.

Information retrieval. To obtain an initial mapping between the model and the textual sub-model descriptions, we apply a statistical technique from information retrieval research: Latent Semantic Analysis (LSA) [LFL98]. For a query (e.g., a sub-model description) over a fixed set of documents (e.g., a set of model element names), LSA scores the relevance of each document to the input query using a vectorization technique.

Model crawling. To create the splitting suggestion, we use the model elements scored highest by LSA as *seeds*. We score each model element's relevance for each target sub-model by traversing the model in breadth-first order for each set of seeds. We calculate the scores of newly accessed model elements using a scoring formula inspired by [Rob05]. Each model element is assigned to the sub-model it was deemed most relevant for.

Tool support. We have implemented the outlined technique in an open-source Eclipse plug-in aiming at the splitting of EMF meta-models and UML class models [SLT14]. The user interface comprises a textual editor for assembling the descriptions and a visual editor for reviewing and post-processing the splitting suggestion. In comparison to hand-tailored model splitting, the accuracy of our implementation averaged at 80% in six case studies.

References

[LFL98] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. An Introduction to Latent Semantic Analysis. *Discourse Processes*, (25):259–284, 1998.

[Rob05] Martin P. Robillard. Automatic Generation of Suggestions for Program Investigation. In *Proc. of ESEC/FSE-13*, pages 11–20, 2005.

[SLT14] Daniel Strüber, Michael Lukaszczyk, and Gabriele Taentzer. Tool Support for Model Splitting using Information Retrieval and Model Crawling Techniques. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*. ACM, 2014.

[SRTC14] Daniel Strüber, Julia Rubin, Gabriele Taentzer, and Marsha Chechik. Splitting Models Using Information Retrieval and Model Crawling Techniques. *Fundamental Approaches to Software Engineering*, pages 47–62, 2014.

[STJS13] Daniel Strüber, Gabriele Taentzer, Stefan Jurack, and Tim Schäfer. Towards a distributed modeling process based on composite models. *Fundamental Approaches to Software Engineering*, pages 6–20, 2013.