Tool Support for Model Splitting using Information Retrieval and Model Crawling Techniques

Daniel G. Strüber, Michael Lukaszczyk, Gabriele Taentzer
Philipps-University Marburg
Department for Mathematics and Computer Science
Hans-Meerwein-Str., 35032 Marburg, Germany
{strueber,lukaszcz22,taentzer}@informatik.uni-marburg.de

ABSTRACT

To facilitate the collaboration in large-scale modeling scenarios, it is sometimes advisable to split a model into a set of sub-models that can be maintained and analyzed independently. Existing automated approaches to model splitting, however, suffer from insufficient consideration of the stakeholder's intentions and add a significant overhead for comprehending the created decompositions. We present a new tool that aims to create more informed model decompositions by leveraging existing domain knowledge in the form of textual descriptions. From the user perspective, the tool comprises a textual editor for assembling the descriptions and a visual editor for reviewing and post-processing the generated splitting suggestions. We preliminarily evaluate the tool in a case study involving a real-life model.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: Tools; D.2.8 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

1. INTRODUCTION

As model-driven engineering is applied in ever-greater scenarios ranging over significant spans in time and space, the maintenance obstacles induced by large models increase in urgency. Large models without a proper decomposition are hard to comprehend, to change, to reuse, and to collaborate on. Even in projects where an initial decomposition is tailored with great care, changing requirements may deem it necessary to refactor for a finer-grained or even orthogonal one. As the manual refactoring of large models is non-trivial and expensive, this problem calls for automation.

Earlier automated approaches to model splitting, such as those presented in [7, 12], suggest techniques based on analysis of strongly connected components or clusters, not accounting for the semantics of the split and the intention for performing it. To address this shortcoming, a recent approach proposed in [11] aims to create model decompositions from existing domain knowledge in the form of textual descriptions: The user provides a set of descriptive texts, each describing one sub-model in the target decomposition. From this input, a splitting suggestion is created using a combined information retrieval and topology analysis approach. The descriptions can be assembled from available requirement or documentation artifacts. However, the input set is not required to be complete: In fact, the approach can support the user in *incrementally discovering* sub-model descriptions.

The contribution of this paper is a tool and supporting semi-automated user process for the splitting of meta-models. We have tested it on large meta-models in the magnitude of 100 to 250 classifiers. Design goals targeted by the tool are usability and extensibility for the splitting of instances of arbitrary meta-models. The remainder of this paper is divided as follows: In Section 2, we briefly explain the underlying technique. The user process is shown in Section 3. In Sections 4 and 5, we elaborate on the design goals and implementation. In Section 6, we present a case study to preliminarily evaluate the proposed tool and user process. We discuss related work and conclude in Sections 7 and 8.

2. BACKGROUND

In this section, we give a brief overview on model splitting as perfomed by our tool. A detailed account is found in [11].

The technique, outlined in Fig. 1, takes three input parameters: The model to be split – in the proposed tool, an EMF meta-model –, a set of textual descriptions of each target sub-model, and a completeness condition. The completeness condition specifies whether the set of sub-model descriptions is complete or partial. The technique creates a set of mappings from model elements to sub-models, calling it *splitting suggestion*. In the case of a complete input set, each element

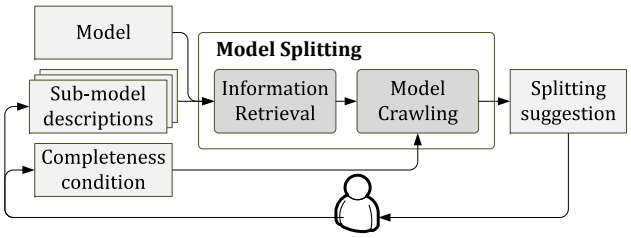

Figure 1: Underlying model splitting technique.

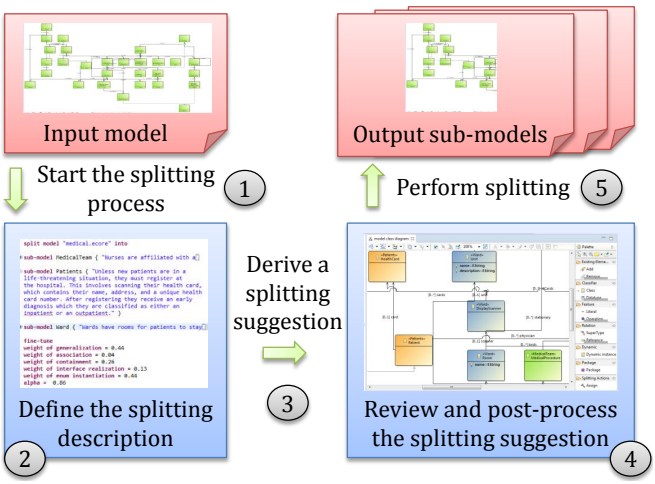

Figure 2: Overview.

```
split model "medical.ecore" into

sub-model MedicalTeam { "Nurses are affiliated with a

sub-model Patients { "Unless new patients are in a
life-threatening situation, they must register at
the hospital. This involves scanning their health card,
which contains their name, address, and a unique health
card number. After registering they receive an early
diagnosis which they are classified as either an
inpatient or an outpatient." }

sub-model Ward { "Wards have rooms for patients to stay

fine-tune
weight of generalization = 0.44
weight of association = 0.04
weight of containment = 0.26
weight of interface realization = 0.13
weight of enum instantiation = 0.44
alpha =  0.86
```

Figure 3: Defining the splitting description.

is assigned to one sub-model. In the partial case, some elements may remain unassigned. The user can inspect the unassigned elements to discover additional sub-models and describe them, incrementally creating a complete split.

Information retrieval. To obtain an initial mapping between the model and the textual sub-model descriptions, we apply an established statistical technique from information retrieval research: Latent Static Analysis (LSA) [8]. 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. To compute the scores, queries and documents are represented as vectors and the similarity between the query vector and each document vector is computed – intuitively speaking, the degree in that they point in the same direction. Mathematically, this is calculated in terms of the cosine, yielding a score between 0 and 1. The vector representation is based on a metrics called *term frequency-inverse document frequency (td–idf)*.

Model crawling. To create the splitting suggestion, we use the model elements ranked highest by LSA as *seeds*. Starting from these seeds, we crawl the model exhaustively to score each model element's relevance for each target sub-model. Afterwards, each model element is assigned to the sub-model it was deemed most relevant for, ties being broken randomly. Model crawling extends an approach proposed in [9]. The underlying intuition is that of a breadth-first search: We first visit and score the seeds' neighbours, then the neighbours' neighbours, et cetera. Scores of newly accessed elements are calculated based on the scores of previously scored elements. The scoring formula accounts for topological properties, such as the connectivity of newly accessed elements, and semantic implications of the respective relationship types (e.g., in meta-models, containment suggests strong connectivity).

3. USER PROCESS

The user process, shown in Fig. 2, comprises two manual tasks (2 and 4) and three automated tasks (1, 3 and 5). The manual tasks rely on human intelligence and domain knowledge. They are facilitated by textual and visual tool support. The automated tasks are triggered by context menu entries.

(1) Start the splitting process. Using a context menu entry on the meta-model to be split, the user triggers the creation of a splitting description file. The splitting description is automatically opened in a textual editor, shown in Fig. 3. By default, the file contains a small usage example.

(2) Define the splitting description. Using the editor, the user assembles the descriptions of the target sub-models. For a comfortable user experience, the editor provides syntax highlighting, static validation, and folding capabilities. The textual editor is also used for configuration: Adding the keyword `partially` and defining a numerical threshold, the user can set the completeness condition in order to obtain a partial split. Furthermore, the user can fine-tune internal parameters used during the execution of the underlying technique. In Fig. 3, the weights assigned to different relationship types and the *alpha* exponent that shapes the scoring function are modified. However, parameter tuning is an optional feature: In [11], we identified a default combination of parameter values that, when applied to six independent class models, achieved an average accuracy of 80% in comparison to hand-tailored decompositions.

(3) Derive a splitting suggestion. Using a context menu entry on the splitting description file, the user triggers the automated creation of a splitting suggestion. A splitting suggestion comprises a set of assignment entries, each holding a link to a model element, a link to a target sub-model, and the relevance score. To compute the splitting suggestion, the technique outlined in Section 2 is applied. The splitting suggestion is persisted to the file system.

(4) Review and post-process the suggestion. To obtain visual access to the splitting suggestion, the user can now open the model in a model editor. The user activates a dedicated layer called *model splitting*. This action triggers the color-coding of model elements corresponding to the splitting suggestion, shown in Fig. 4. As further visual aid, the assignment of a model element is also displayed textually above its name. For post-processing, the user may want to change some assignments for model elements that were not assigned to the proper target sub-model. This is done using

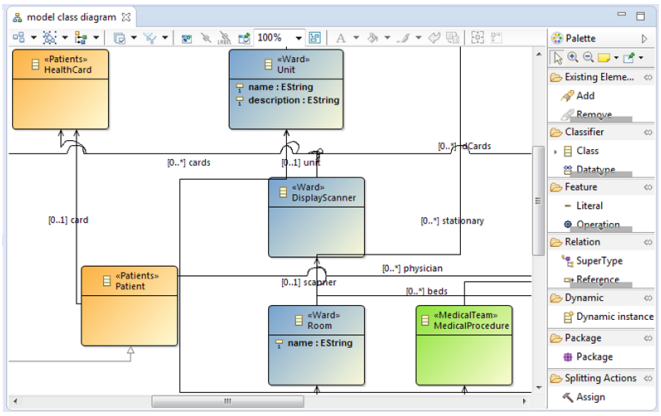

Figure 4: Reviewing and post-processing the splitting suggestion.

the palette tool entry *Assign*. When the user reassigns a model element, the respective entry in the splitting suggestion is automatically updated. It is worth mentioning that if the user is not satisfied with the results, he or she may iterate Steps 2 to 4 as often as required, tweaking the descriptions and parameter settings. One important scenario for this is the discovery of new sub-models: The user can set the completeness condition to *partial* in Step 2 which leads to some model elements not being assigned in Step 3. The user inspects these elements in Step 4 to create new sub-model descriptions.

(5) Perform splitting. Given that the user is satisfied with the post-processed splitting suggestion, the actual splitting can be triggered by the user. The user may choose from two context menu entries: One for splitting the input model into multiple physical resources, the other for splitting it into sub-packages within the same resource.

4. DESIGN GOALS

In this section, we shortly discuss design goals that were fundamental in the design of the proposed tool.

Extensibility. The underlying technique possesses an innate extensibility that should be carried over to the end-user. It is applicable to models conforming to arbitrary meta-models, given that they fulfill two properties: (i) Model elements must have meaningful textual descriptions that a splitting description can be matched against. (ii) Except for trivial reconciliation, constraints imposed by the meta-model may not be broken in arbitrary sub-models. We address this design goal by using a framework approach: To customize the tool for a new meta-model, the user subclasses a set of base classes. For instance, to define how input models are converted to a generic graph representation used during crawling, they subclass a class named *GraphBuilder*.

Usability. The design of the tool is informed by *Cognitive Dimensions*, a framework for the human-centered design of languages and tools [6]: Providing an editable visual layer on top of a standard editor is a major step towards *visibility* – visual accessibility of elements and their relations – and away from *viscosity* – resistance to change. *Closeness of mapping* is implemented by a domain-specific language for splitting descriptions with custom editor support. *Premature commitment* is inhibited and *progressive evaluation* is promoted by providing an incremental process that allows tweaking with input values while receiving rapid feedback. For *traceability*, our file-based approach to user input allows to keep the splitting description and use it later, e.g., for documentation purposes.

5. IMPLEMENTATION

Eclipse Modeling Framework [10] is the de-facto reference implementation of the EMOF modeling standard. Consequently, it was natural for us to design the new tool as an extension for EMF. As such, it can be plugged into an existing Eclipse installation without further effort. For the splitting description editor, we leveraged the powerful code generation facilities of Xtext [5]. We defined a simple domain-specific language for splitting descriptions. The editor with its syntax highlighting and code completion features was fully generated by Xtext. For customization, we added a couple of checks (e.g., forbidden characters, uniqueness of sub-model names). The visual splitting layer is an extension of EcoreTools 2.0 [2] which is based on the Sirius [4] framework and, as of June 2014, determined to be part of the new Eclipse release Luna 4.4. We used this new technology as we benefit from its support for multiple viewpoints, allowing us to tailor a *splitting viewpoint* to our needs.

6. CASE STUDY

In a case study, we investigated two research questions: (RQ1) How efficient is the proposed tool in comparison to manual splitting? (RQ2) Is the proposed tool usable?

6.1 Subjects and Task

Subject model: *Extended Joomla-Specific Language (eJSL)* is a meta-model for web applications based on the Joomla content management system [3]. It comprises 116 classes, 39 enumerations, 176 enumerated attributes, 41 generalizations, 145 containment references, and 47 plain references. eJSL was designed by a doctoral student affiliated with our research group we shall refer to as X. X has significant experience in modeling language design. Previous to our work, X manually split eJSL into five sub-models, calling them *Pages*, *Content*, *Menu*, *User*, and *Configuration*. According to his account, he invested a significant effort that spanned, among other duties, over the course of two weeks. He printed the diagram on paper, cut and reassembled fragments. Afterwards, he assigned colors to model elements in the diagram editor and layouted them by hand.

Task: We instructed another software engineer, referred to as Y, to decompose eJSL using the tool. Y is a doctoral student with significant experience in modeling language design, but unrelated to eJSL and model splitting. We asked X to provide the required domain knowledge in the form of descriptive texts briefly explaining his intuitions for the hand-tailored decomposition. The descriptions, each consisting of 85 words on average, were handed to Y in a text document. The task given to Y was to create *a decomposition that faithfully reflects the separation of concerns proposed by the textual descriptions*. We briefly instructed Y in the usage of the tool based on the example shown in Fig. 3 and 4 and encouraged him to make use of post-processing.

6.2 Results

Efficiency. To approach (RQ1), we define *efficient* as *requiring a minimal amount of time to create an accurate result*. Positing the hand-tailored split as perfectly accurate, we measured accuracy of the tool-supported split in terms of average F-measure, considering both precision and recall. Accuracy was determined before and after post-processing: During review of the initial splitting suggestion *S1*, Y reassigned five model elements to create the final suggestion *S2*. From S1 to S2, precision increased from 82% to 86% and recall from 84% to 88%, determining a rise in F-measure from 83% to 87%. It took Y five minutes to create *S1*. The reviewing and post-processing that brought the 4% gain took further 55 minutes. Consequently, in terms of extrapolated overall amount of time, tool-supported splitting outperformed manual splitting.

Usability. To approach (RQ2), we conducted an informal interview. Y perceived the user process as comprehensible, the description editor as easy to use and the color-coding as useful. An activity found crucial during post-processing was examining the direct neighbours of a model element. Y perceived this task as cumbersome: He often had to navigate for edge targets outside the visible scope. For future work, we aim at dedicated support for this activity: On selection, neighbourhood information should be instantly available in a tool-tip displaying the names of adjacent elements. One further suggestion by Y, the color-coding of edges, directly made it into the current version. Y also invested considerable time in layouting, i.e., aligning the color-coded model elements into groups – an activity outside of the scope of this work. It is an interesting challenge to devise a layouting algorithm that aligns the sub-models of a model as clusters. Inspection of the false positives and negatives in S2 revealed that 50% of them concerned enumerations, the other 50% concerning classes. Y pointed out that enumerations were hard to relate to classes visually as they are not connected by edges. We consider representing enumerated attributes as edges rather than class members in future work.

6.3 Validity

Threats to external validity – or generalizability – are the size of the input model and the size of the test group. It remains a question left to future work whether our tool scales for meta-models of significantly more elements. However, an analysis of publicly available meta-models[1] indicates the input model size to be typical for large meta-models demanding an adequate decomposition. The test group size indeed precludes claims for generality, but allows to provide tentative evidence for critical design weaknesses and benefits. A potential threat to internal validity – or freeness from systematic error – is the flow of information from the control group to the test group. To mitigate this threat, we ascertained in consultation with X that the textual descriptions in vagueness and level of detail represented the intuitions for splitting *before* the manual split was executed.

7. RELATED WORK

In this section, we discuss related tooling. A survey of work related to the underlying *approach* is provided in [11].

[1]http://www.emn.fr/z-info/atlanmod/index.php/Ecore

In the Democles model composer [1], the user can iterate the lattice of all permitted decompositions by unfolding entries in a tree-like wizard. Graphical presentation of a split is provided by an add-on graph visualization library. However, this visualization is read-only and not integrated with a modeling editor, ruling out the re-assigning of model elements for post-processing as supported by the new tool.

The splitting tool proposed in [12] makes classic clustering algorithms available for EMF models. It provides a wizard for the selection and customization of algorithms. However, except for numerical input parameters, the user cannot influence the generated results. The tool provides a tree-based editor for the reassigning of model elements to target submodels, but does not present any visual feedback.

8. CONCLUSION

In this paper, we present a tool for the splitting of large meta-models. The tool provides a textual editor that allows defining the desired target sub-models by means of textual descriptions. It generates a splitting suggestion that can be reviewed and post-processed in a visual editor. Based on the splitting suggestion, the input model can be automatically split either into multiple resources or packages within one resource. The tool is open source and can be found, along with the models mentioned in this paper, at `https://www.uni-marburg.de/fb12/swt/forschung/software`. In the future, we plan to apply the technique on other models than class models, deeming it necessary to account for constraints.

9. REFERENCES

[1] Democles. http://democles.lassy.uni.lu/, May 2011.

[2] Ecoretools 2.0. http://www.eclipse.org/ecoretools/, May 2014.

[3] Joomla. http://www.joomla.org/, May 2014.

[4] Sirius. http://www.eclipse.org/sirius/, May 2014.

[5] Xtext. http://www.eclipse.org/xtext/, May 2014.

[6] T. R. G. Green and M. Petre. Usability analysis of visual programming environments: a cognitive dimensions framework. *Journal of Visual Languages & Computing*, 7(2):131–174, 1996.

[7] P. Kelsen, Q. Ma, and C. Glodt. Models within models: Taming model complexity using the sub-model lattice. *Fundamental Approaches to Software Engineering*, pages 171–185, 2011.

[8] T. K. Landauer, P. W. Foltz, and D. Laham. An Introduction to Latent Semantic Analysis. *Discourse Processes*, (25):259–284, 1998.

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

[10] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: Eclipse Modeling Framework*. Pearson Education, 2008.

[11] D. Strüber, J. Rubin, G. Taentzer, and M. Chechik. Splitting models using information retrieval and model crawling techniques. *Fundamental Approaches to Software Engineering*, pages 47–62, 2014.

[12] D. Strüber, M. Selter, and G. Taentzer. Tool support for clustering large meta-models. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*, page 7. ACM, 2013.