Managing Model and Meta-Model Components with Export and Import Interfaces

Daniel Strüber, Stefan Jurack, Tim Schäfer, Stefan Schulz, Gabriele Taentzer

Philipps-Universität Marburg, Germany,
{strueber,sjurack,timschaefer,schulzs,taentzer}
@informatik.uni-marburg.de

Abstract. *Composite modeling* provides a modularity mechanism for models. To facilitate information hiding, it supports the declaration of export and import interfaces at the meta-model and model levels. In this paper, we present a tool set of wizards and editor extensions that makes composite modeling available to developers. The tool set is based on the Eclipse Modeling Framework. We demonstrate its use during the model-driven management of data-oriented applications.

1 Introduction

As the requirements imposed on Model-Driven Engineering (MDE) tools and techniques grow in complexity, so do the involved models, rendering their maintenance, transformation, and overall management highly challenging. To support developers during these tasks, appropriate modularity mechanisms are required [9]. Modularity is a fundamental software engineering concept with well-established principles that should ideally inform the design of such mechanisms. Two main principles are: (i) establishing a *separation of concerns* into distinct modules, (ii) facilitating *information hiding* and restricting *visibility* between modules by means of explicit interfaces.

In the MDE community, the Eclipse Modeling Framework (EMF) [12] is a widely used base technology. In EMF, a separation of concerns can be established by distributing information over a set of related models. Consider the left part of Fig. 1 for a pair of data models from operational systems for a tourism agency and an airline. The tourism agency model contains a trip assigned to a flight from the airline model. The dashed arrow denotes a *remote reference*: When the travel agency system loads the model, the flight is represented as a proxy object. In the case of data accesses on the flight, the flight model is loaded and added to the memory representation of travel model (right part). All model contents, including critical data such as flight logs and pilots, become visible.

This example highlights a lack of information hiding at the model level in EMF, in this case resulting in a situation that may compromise security and privacy. Other affected concerns include performance, analyzability, and collaborative development: During queries and transformations, considering a large set of models in union may be inefficient. Since models are not self-contained units, checking static properties of the individual models is prohibited unless certain restrictions are imposed [1]. A main issue during collaboration is proneness to inconsistencies. As an example, consider a situation where a developer deletes a model element from a model, unaware that this element is referred to from another model. This deletion results in a broken model reference.

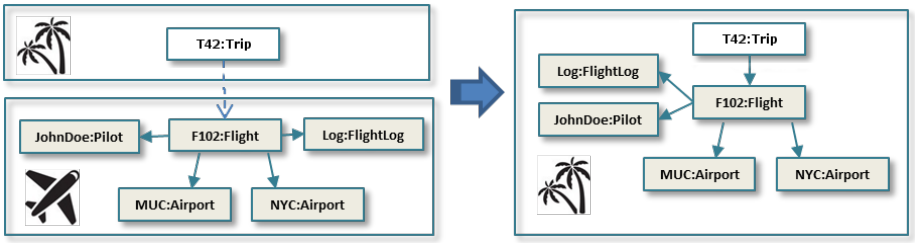

Fig. 1: Related models in EMF: *before* and *after* proxy resolution.

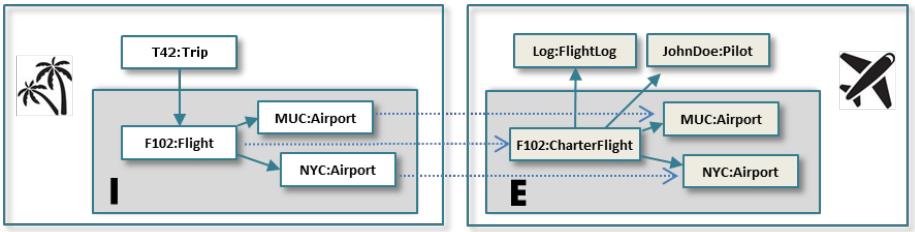

Fig. 2: Two model components with export and import interfaces.

A modularity mechanism addressing these issues is provided by *composite modeling* [6,16]. A composite model is a set of *components* where each component comprises a model with a set of export and import interfaces. Export and import interfaces declare subsets of model elements offered to and obtained from the environment. In Fig. 2, the travel agency has an import interface; the airline component has an export interface. Each model element in the import interface is mapped to a corresponding export element (dotted lines). In comparison to Fig. 1, the travel agency component now maintains the flight and its assigned airports as autonomous, but distinguished objects. We refer to these objects as *delegate objects*. Similar to a proxy, a delegate object represents an object stored somewhere else. But despite reflecting some of the remote object's features, a delegate object has an own identity. Consequently, as shown in our earlier work [16], components are self-contained units amenable to analysis and collaborative editing.

In this paper, we introduce a tool set that makes composite models available to model and meta-model developers. The tool set provides an implementation of composite models that is generic in the sense that it is applicable to any EMF-based host language. It comprises dedicated wizards plus a set of extensions of a visual meta-model editor, a generic tree-based model editor, and a model transformation tool. We provide these tools at `http://www.informatik.uni-marburg.de/~swt/compoemf/`.

2 Overview

Consider Fig. 3 for an overview of our tool set. As a starting point, we assume a set of related meta-models, e.g., for interacting systems or views on the same system. If these meta-models have not been developed as a components from scratch, they need to be extended with export and import interfaces (step 1). The resulting components can be

subject to editing (step 2). Once their development converges, the meta-model components are instantiated by model components (step 3). These components can be queried and transformed in the same manner as models in typical MDE scenarios (step 4).

Fig. 4 shows two application meta-models used to specify travel agency and airline operative systems; colors and stereotypes can be temporarily ignored. While these meta-models share a subset of common elements, such as classes `Flight` and `Airport` and their attributes, the specifications differ in the level of detail: The airline meta-model contains subclasses for charter and scheduled flights as well as additional attributes for airports. In our approach, we address such mismatches by allowing individual references and attributes to be exported and imported, and generalization to be flattened.

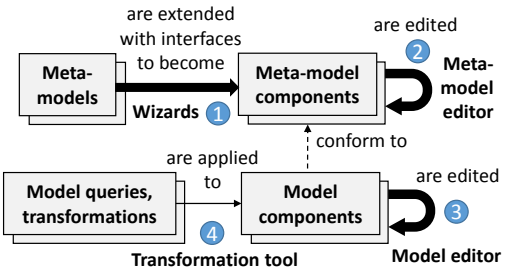

Fig. 3: Overview.

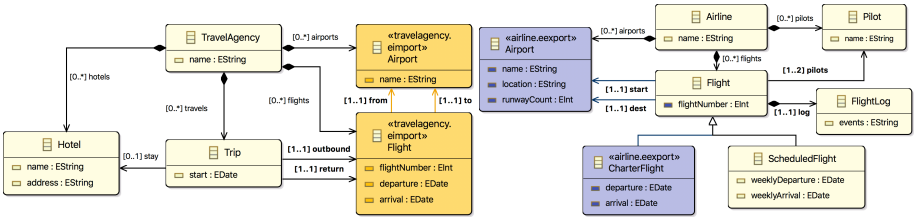

Fig. 4: Meta-models for travel agency (left half) and airline systems (right half).

(1) Extend meta-models. We provide a set of wizards to introduce export and import interfaces in a set of meta-models. Our wizards, shown in Fig. 5, allow the specification of a set of classes and features from the input model. A selection of classes and features to be exported or imported is specified using check-boxes. In the case of import interfaces, corresponding classes from an export interface have to be selected. A mapping from import to export elements is automatically derived using a name-based heuristics; missing mappings can be set by using either drag and drop or buttons.

(2) Edit meta-model components. The meta-model components can be edited using an extension of EMF's meta-model editor, the main part being shown in Fig. 4. The assignment of classes, references, and attributes to interfaces is denoted visually, using stereotypes and a custom color scheme. In addition, the editor provides dedicated functionality for the manipulation of interfaces, e.g., the (re)assigning of elements or the creation of new additional interfaces from a selection of elements. In our implementation of these features, we harnessed the view management of Sirius (`http://www.eclipse.org/sirius`) to extend the EMF meta-model editor.

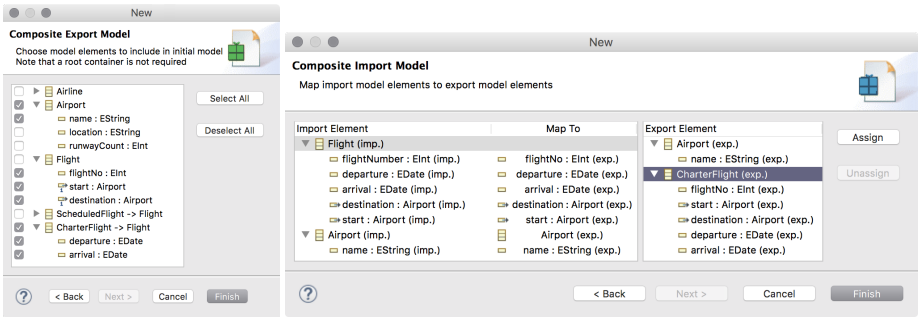

Fig. 5: Export and import creation wizards.

(3) Edit model components. The meta-models can now be instantiated. Export and import relationships can be edited using our extension of EMF's generic tree-based editor as shown in Fig. 6. Export and import interfaces, denoted *E* and *I*, are displayed as child nodes of their body models *(B)*. Their included elements are shown as distinct nodes; to help users trace their relationships, all elements related to the currently select one are highlighted. To edit the interfaces between models in an integrated visual representation (see Fig. 2), a customization of the involved editors is required. In our ongoing work, we develop a framework that allows to reduce the editor customization overhead. The Sirius view management is a promising prerequisite for this framework.

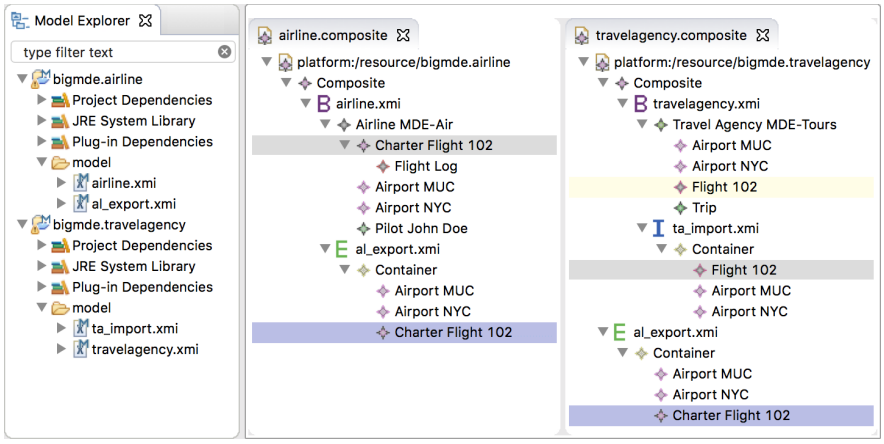

Fig. 6: Tree-based editors in EMF.

(4) Apply queries and transformations. Existing model query and transformation tools do not provide dedicated support for interfaces. Therefore, queries and transformations cannot consider export-import relations between models. To this end, we provide CompoHenshin, a model transformation tool on top of Henshin, a graph-based model transformation tool based on EMF. CompoHenshin comprises a visual editor for the

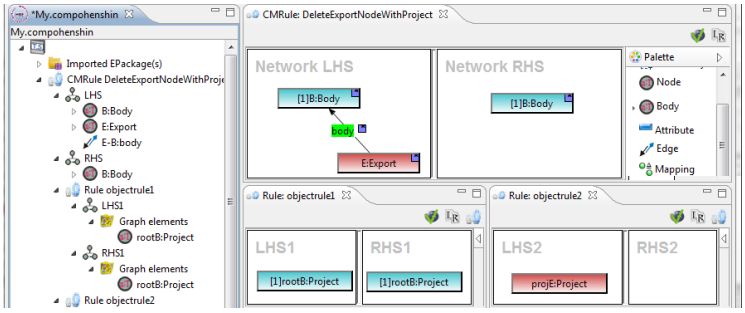

Fig. 7: CompoHenshin editor.

specification of composite rules [7] that can be applied to a set of model components. A composite rule (see Fig. 7) comprises a network rule and a set of object rules. Each object rule specifies the local change for one of the nodes in the overall set of models.

3 Discussion

Limitations. To support interoperability with existing tools, it is desirable to augment meta-models and models in a transparent manner: Export and import interfaces should be added on top of existing models, rather than changing them. In our approach, this is only possible if the meta-models contain suitable reference classes, such as `Flight` and `Airport` in our example. Otherwise, the meta-models need to be changed to contain such classes. The potential performance savings during analysis and transformations largely depend on the considered scenario. The most visible gain is to be expected if the component of interest is small, while its context is huge. For example, an OCL constraint can be expressed as a small model (the abstract syntax of the constraint) referencing a much larger one (the OCL standard library with its operators and literals).
Related work. Kelsen and Ma propose a black-box model modularity mechanism based on the fragmentation of a meta-model along some of its associations [8]. The approach by Heidenreich et al. [5] uses a component model and a composition language to express components, interfaces, and composition steps. These works assume a mandatory "weaving" step, whereas in our approach imported elements are self-contained objects reflecting features of a remote counterpart. Other works consider interfaces at the meta-model level only [17,18] or for particular DSLs [2]. Amálio et al. [1] provide a modularity approach based on *proxy nodes*, a concept emulating EMF's proxy mechanism.

A related line of work considers the splitting of a large model into multiple parts. Garmendia et al. [3] propose a tool to introduce a package structure in a model based on annotations in the meta-model. This tool can be used to explore large models [4]. Scheidgen et al. [10,11] provide a technique and tool for the fragmentation of large models for fast persistence and loading. The technique is based on annotating fragmentation points in the underlying meta-model. In our own work, we have applied clustering to optimize cohesion during splitting [13] and information retrieval techniques to consider the user intention during splitting by retrieving it from given text documents [14,15].

4 Conclusion

We have proposed a tool set that facilitates *information hiding* at the level of metamodels and models. The distinguishing feature of our tool set is that model elements imported from somewhere else are managed as distinct objects with their own identity. By maintaining modules as self-contained units, our approach may facilitate efficient static analysis, queries, and transformations, and developer independence during collaboration. In the future, we intend to evaluate this conjecture on large realistic models.

References

1. Amálio, N., de Lara, J., Guerra, E.: Fragmenta: A theory of fragmentation for MDE. In: Int. Conf. on Model Driven Engineering, Languages and Systems. pp. 106–115. IEEE (2015)
2. Arifulina, S., Mohr, F., Engels, G., Platenius, M.C., Schafer, W.: Market-specific service compositions: Specification and matching. In: Services (SERVICES), 2015 IEEE World Congress on. pp. 333–340. IEEE (2015)
3. Garmendia, A., Guerra, E., Kolovos, D.S., de Lara, J.: EMF Splitter: A Structured Approach to EMF Modularity. Workshop on Extreme Modeling pp. 22–31 (2014)
4. Garmendia, A., Jiménez-Pastor, A., de Lara, J.: Scalable model exploration through abstraction and fragmentation strategies. In: BigMDE Workshop on Scalability in Model Driven Engineering. pp. 21–31. CEUR (2015)
5. Heidenreich, F., Henriksson, J., Johannes, J., Zschaler, S.: On Language-Independent Model Modularisation. T. Aspect-Oriented Software Development VI pp. 39–82 (2009)
6. Jurack, S., Taentzer, G.: Towards Composite Model Transformations Using Distributed Graph Transformation Concepts. In: Schürr, A., Selic, B. (eds.) Int. Conf. on Model Driven Engineering Languages and Systems. pp. 226–240. Springer (2009)
7. Jurack, S., Taentzer, G.: Transformation of Typed Composite Graphs with Inheritance and Containment Structures. Fundam. Inform. 118(1-2), 97–134 (2012)
8. Kelsen, P., Ma, Q.: A Modular Model Composition Technique. In: Int. Conf. on Fundamental Approaches to Software Engineering. pp. 173–187. Springer (2010)
9. Kolovos, D.S., Rose, L.M., Matragkas, N., Paige, R.F., Guerra, E., Cuadrado, J.S., De Lara, J., Ráth, I., Varró, D., Tisi, M., Cabot, J.: A Research Roadmap towards Achieving Scalability in Model Driven Engineering. In: BigMDE Workshop on Scalability in Model Driven Engineering. p. 2. ACM (2013)
10. Scheidgen, M.: Reference representation techniques for large models. In: BigMDE Workshop on Scalability in Model Driven Engineering. pp. 5:1–9. ACM (2013)
11. Scheidgen, M., Zubow, A., Fischer, J., Kolbe, T.H.: Automated and transparent model fragmentation for persisting large models. In: Int. Conf. on Model Driven Engineering Languages and Systems. pp. 102–118. Springer (2012)
12. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Pearson Education (2008)
13. Strüber, D., Lukaszczyk, M., Taentzer, G.: Tool support for model splitting using information retrieval and model crawling techniques. In: BigMDE Workshop on Scalability in Model Driven Engineering. pp. 44–47. CEUR (2014)
14. Strüber, D., Rubin, J., Taentzer, G., Chechik, M.: Splitting models using information retrieval and model crawling techniques. In: Int. Conf. on Fundamental Approaches to Software Engineering, pp. 47–62. Springer (2014)
15. Strüber, D., Selter, M., Taentzer, G.: Tool support for clustering large meta-models. In: BigMDE Workshop on Scalability in Model Driven Engineering. pp. 7:1–4. ACM (2013)
16. Strüber, D., Taentzer, G., Jurack, S., Schäfer, T.: Towards a distributed modeling process based on composite models. In: Fundamental Approaches to Software Engineering, pp. 6–20. Springer (2013)
17. Weisemöller, I., Schürr, A.: Formal definition of MOF 2.0 metamodel components and composition. In: Model Driven Engineering Languages and Systems, pp. 386–400. Springer (2008)
18. Živković, S., Karagiannis, D.: Towards metamodelling-in-the-large: Interface-based composition for modular metamodel development. In: Enterprise, Business-Process and Information Systems Modeling, pp. 413–428. Springer (2015)