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. When developing software in distributed teams, the maintenance and comprehension of models require an adequate modularization concept. Existing approaches that support the modularization of models over distributed resources demand a compile-time or load-time integration step of uniting the involved modules to form one big model. In our earlier work, we have proposed composite models, a component-oriented modularization approach for models that facilitates fundamental engineering principles such as encapsulation, decoupling, and information hiding at the model level as well as the meta-model level. In this paper, we present an implementation of composite models. Our implementation comprises a tool set of wizards and editor extensions based on the Eclipse Modeling Framework. We demonstrate our tool set in the use-case scenario of developing a data-oriented applications in a model-driven manner.

Keywords: composite models, model modularization, distributed modeling, EMF

1 Introduction

Model-Driven Engineering (MDE) is now a common practice in many software domains. As requirements grow in size and complexity, so do the resulting models, leading to a need for appropriate modularization facilities to support maintainability and comprehension [9]. The notion of modular modeling is inspired by that of modular programming as introduced in the seminal paper by David Parnas [10]. The key properties of modular programming are: (i) the *identification and separation of distinct concerns* that are distributed over a set of modules and (ii) the identification of interfaces establishing *encapsulation* and restricting *visibility* between modules.

In the MDE community, Eclipse Modeling Framework (EMF) [13] 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 travel 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.

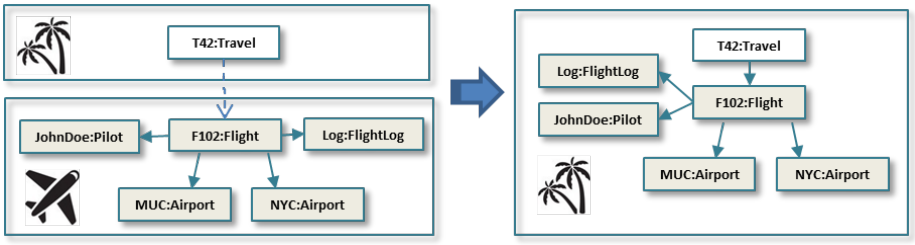

Fig. 1: Related models in EMF: *before* and *after* proxy resolution.

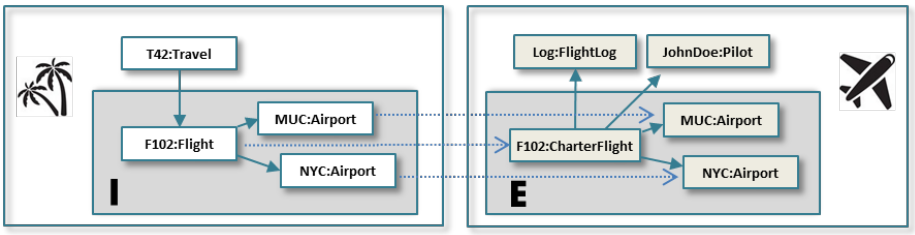

Fig. 2: Two model components with export and import interfaces.

This example highlights a security issue related to the remote reference mechanism. Other affected concerns include performance, analyzability, and collaborative work: During queries and transformations, considering a large model set in union may be inefficient. Checking static properties of the individual models is prohibited unless certain restrictions are imposed [1], since models are not self-contained units. A main issue during collaboration is proneness to inconsistencies. As an example, consider a situation where a developer deletes a model element from their model, unaware that this element is referred to from another model. This deletion results in a broken model reference.

A modularization technique addressing theses issues is provided by *composite models* [6,17]. 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 [17], components are self-contained units amenable to analysis and collaborative editing.

In this paper, we introduce a basic tool set that makes composite modeling available to model and meta-model developers. The tool set provides an implementation of composite models that is generic (i.e., applicable to any EMF-based host language) and transparent (i.e., allows reusing existing meta-models).

2 A core tool set for composite models

We give an overview of our tool set and its supported features in Fig. 3. As a starting point, we assume a set of meta-models used to describe related domains in a MDE scenario. These meta-models are turned into meta-model components by extending them with export and import interfaces (step 1). The resulting components can be subject to additional editing (step 2). Once their development converges, the meta-model components are instantiated by model components with export and import interfaces (step 3). These components can be queried and transformed in the same manner as regular models in typical MDE scenarios (step 4).

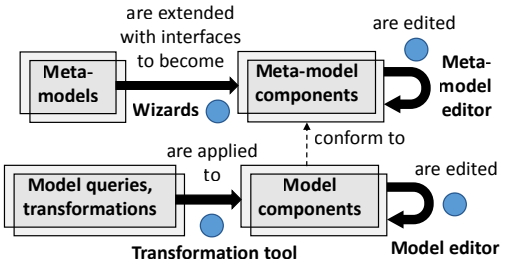

Fig. 3: Overview.

Consider the two application meta-models for travel agency and an airline operative systems shown in Fig. 4. While sharing the classes `Flight` and `Airport`, the specification differs in the level of detail: The airline meta-model specifies subclasses for the charter and scheduled flight cases as well as additional attributes for the airport class. In our approach, we address such mismatch situations by allowing the exporting and importing of individual features as well as the flattening of generalizations.

(1) Extend meta-models. We provide a set of wizards to introduce export and import interfaces in a set of meta-models. Export and import interfaces specify model parts provided to and obtained from the environment. Each element contained in an import interface refers to an exported element in another component. Interfaces of meta-model components specify potential export-import relations between model components.

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. This mapping from import to export classes can be set using either drag and drop functionality or buttons.

(2) Edit meta-model components. The meta-model components can be edited using an extension of EMF's meta-model editor, excerpts being shown in Fig. 6. The assignment of classes, references, and attributes to interfaces is highlighted visually,

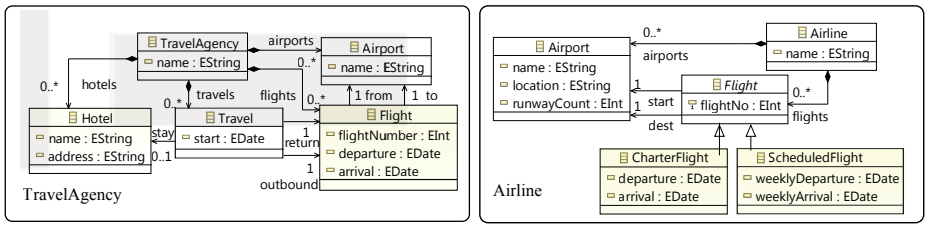

Fig. 4: TravelAgency and Airline meta-models.

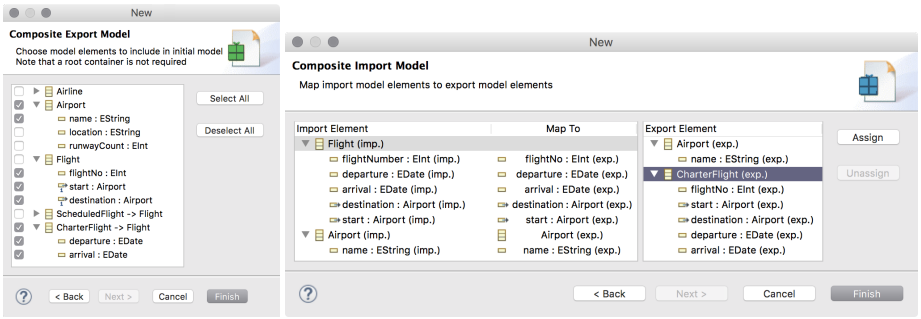

Fig. 5: Export and import creation wizards

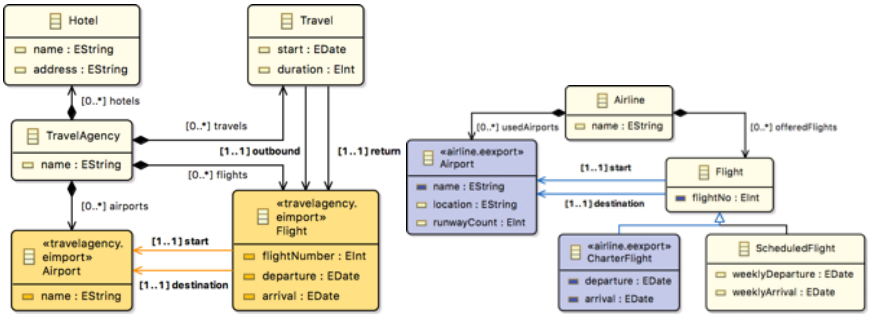

Fig. 6: Visualizing interfaces in the EcoreTools meta-model editor.

using a view on top of the default editor. 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.

(3) Instantiate models. The meta-models can now be instantiated. Export and import relations between models can be edited using an extension of the generic tree-based model editor (not shown). To edit the interfaces between models visually, in the style of Fig. 2, a customization of the involved editors is required. In our ongoing work, we are developing a generic framework that allows to reduce this overhead for editor customization. A key prerequisite for such a framework is the *view* management provided by Sirius, a concept we harnessed to develop our meta-model editor extension.

(4) Apply queries and transformations. Existing model query and transformation tools do not provide dedicated support for interfaces. Therefore, queries and transformations cannot consider the export-import relations between models. To support support these tasks, we provide a dedicated model transformation tool on top of Henshin, a graph-based model transformation language for EMF models. The tool comprises a visual editor (not shown) for the specification of composite transformation rules [7] that can be applied to a set of model components.

3 Related work

Several works have investigated information hiding at model level. A black-box model modularization approach based on the fragmentation of a meta-model along some of its associations, called fragmentation edges, is proposed in [8]. The authors distinguish two types of modularization artifacts: modules as self-contained units and fragments as fragmentary units that contain fragmentation edges. Fragments and modules are brought to life in an obligatory weaving phase. The approach by Heidenreich et al. [5] comprises a component model and a composition language. The component model describes components with explicit export and import interfaces. The language gives a syntax for expressing interfaces and composition steps. The key distinction between these works and ours is that they assume a composition or integration step, whereas in our approach imported elements are self-contained objects that, supported by a delegation technique, reflect the features of remote objects. Amálio et al. [1] provide a modularization approach based on model fragments being related through *proxy nodes*, a mechanism emulating the one provided by EMF. Certain global properties of a modularized model can be checked locally, assuming that suitable constraints are met.

Several works provide interfaces at the meta-model level to improve the development of meta-models [18,19]. These works do not address the model level explicitly. Conversely, another class of approaches is based on defining interfaces for one particular domain-specific language, e.g., service specifications in the case of [2].

Separation of concerns at model level has been addressed by a line of work on the splitting of a large model into multiple parts. Garmendia et al. [3] propose a tool to introduce a package structure in a large model based on annotations in the meta-model. This tool has been used in a process aimed at the exploration of large models [4]. Scheidgen et al. [11,12] provide a technique and tool for the fragmentation of large models for faster persistence and loading. The technique is based on annotating fragmentation points in the underlying meta-model. In our own work, we have applied standard clustering techniques to optimize for high cohesion during splitting [14]. We have used information retrieval techniques to extract the user intention during splitting from text documents, such as requirements specifications or the project documentation [15,16].

4 Conclusion

In this paper, we propose a tool set that facilitates the modularization of meta-models and models. By allowing to augment a set of meta-models and models with export and import interfaces, the tool set provides support for the *separation of concerns* and *information hiding* principles at meta-model and model level. The distinguishing feature of our approach is that it allows to maintain components as self-contained units: Model elements imported from another model are managed as distinct model elements with their own identity. This approach establishes a loose coupling between components, facilitating developer independence during collaborative editing, local static analysis, and efficient transformations. The tool set comprises dedicated wizards plus a set of extensions of existing an visual meta-model editor, a generic tree-based model editor, and a model transformation tool. We provide these tools online at `http://www.informatik.uni-marburg.de/~swt/compoemf/`.

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 p. 21 (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. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Commun. ACM 15, 1053–1058 (December 1972)
11. Scheidgen, M.: Reference representation techniques for large models. In: BigMDE Workshop on Scalability in Model Driven Engineering. p. 5. ACM (2013)
12. 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)
13. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Pearson Education (2008)
14. 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. Citeseer (2014)
15. 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)
16. Strüber, D., Selter, M., Taentzer, G.: Tool support for clustering large meta-models. In: BigMDE Workshop on Scalability in Model Driven Engineering. p. 7. ACM (2013)
17. Strüber, D., Taentzer, G., Jurack, S., Schäfer, T.: Towards a distributed modeling process based on composite models. In: Int. Conf. of Fundamental Approaches to Software Engineering, pp. 6–20. Springer (2013)
18. 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)
19. 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)