Extending GMF: Generating Domain-Specific Model Editors with Complex Editing Commands

E. Biermann[1], A. Crema[1], K. Ehrig[2], C. Ermel[1], C. Köhler[3], R. Schmutzler[1], and G. Taentzer[4]

[1] Technische Universität Berlin, Germany,
[2] University of Leicester, United Kingdom,
[3] CWI Amsterdam, The Netherlands,
[4] Philipps-Universität Marburg, Germany
`gmftrans@tfs.cs.tu-berlin.de`
`http://tfs.cs.tu-berlin.de/emftrans/gmftrans`

1 Overview

In software system development, domain-specific visual notations are increasingly used and need a tool environment consisting of visual editors, simulators, model transformers, etc. Several Eclipse projects head for a meta technology to define domain-specific modeling languages. The Eclipse Modeling Framework (EMF) [4] can be used to define the underlying models of visual editors. A visual editor may be generated using the *Graphical Modeling Framework (GMF)* [3] which started recently as Eclipse technology subproject aiming at providing an infrastructure for generating visual editors in Eclipse. In essence, GMF forms a bridge between EMF and GEF [2], whereby a diagram definition is linked to a domain model which serves as input to the generation of a visual editor.

GMF-generated editors offer basic editing commands to create, edit, move and delete single model elements (basic editing). Graph transformation-based editors (see e.g. Tiger [6]) show that the generation of editors with complex editing commands is also possible. Editing e.g. activity diagrams, there might be editing commands available which insert or delete a complete decision structure in one step.

In this paper, we describe how meta model-based editor design and generation performed by GMF [3], can be extended by graph transformation concepts to define and generate complex editing commands to be used in GMF-generated visual editors.

2 Extension of the GMF Development Environment

In addition to the *domain model* and the *graphical definition model*, a visual editor for defining complex editing commands is provided, where EMF transformation rules for complex editing commands can be defined based on the domain model. This step is optional. The extended *tooling definition model* is used to define the commands for the editor palette. After having defined all these models separately, the *mapping model* establishes a connection between them and is the input for the generation process. Fig. 1 shows an overview of the design workflow in the extended GMF using a dashboard, where the original GMF workflow is extended by the specification of a *Transformation Rule Model*.

The EMF transformation approach [1] is closely related to algebraic graph transformation: Basically, an EMF transformation is a rule-based modification of an EMF source model resulting in an EMF target model. Both, the EMF source and target models are typed over an EMF core model. All modifications are made in-place, i.e. the source model is not copied before modification. The left window of Fig. 2 shows the visual designer for EMF transformations where the underlying meta model for activity diagrams is depicted at the bottom and one of the transformation rules, i.e. a rule for inserting a start diagram, is shown at the top. A negative application condition (NAC) ensures that this rule is applied only to the empty activity diagram.

After having defined all required editing commands analogously, all those which should show up in the palette have to be identified in the GMF tooling model, and the GMF mapping model is extended by the definition of the transformation model. A suitable set of complex editing commands may ensure the well-formedness of

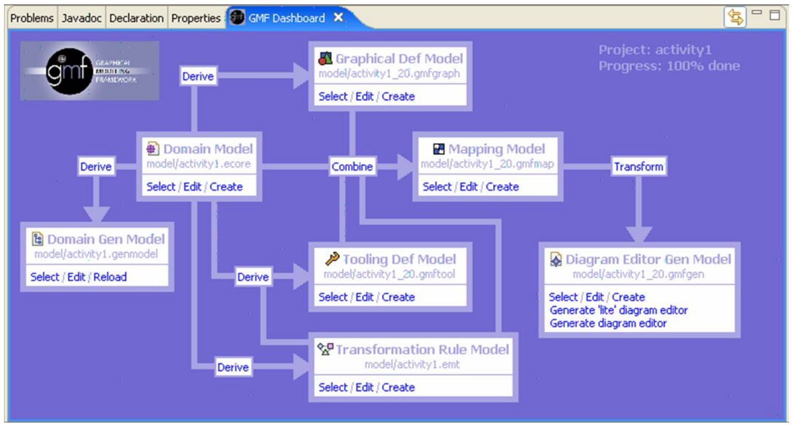

Fig. 1. GMF dashboard extended by transformation rule model for specifying editing commands

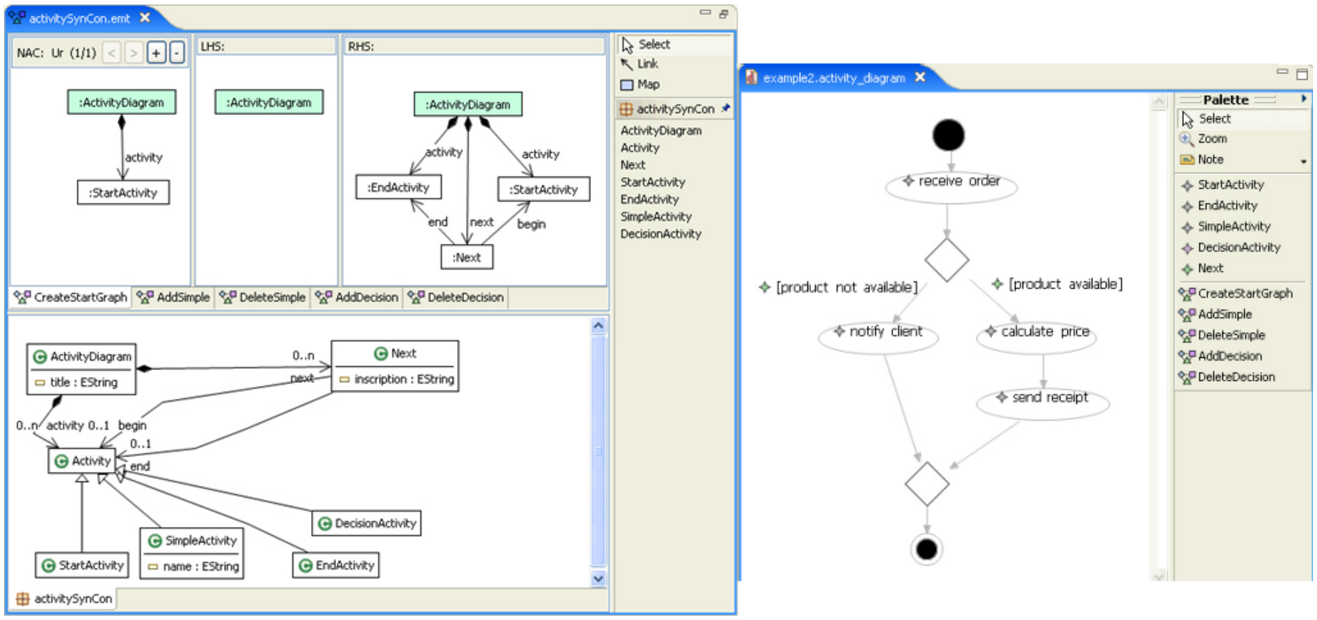

Fig. 2. Tool environment for EMF transformation and generated GMF editor for activity diagrams

diagrams edited in the generated editor, e.g. it is possible to define a variant of activity diagrams allowing only one start and one end activity. A screenshot of a generated editor for activity diagrams with four complex editor commands in the palette is shown in the right window of Fig. 2.

3 Extension of the GMF Runtime Environment

The editor generation process in the extended GMF version results in an editor where default editing operations as well as complex ones may be provided by the palette.

We describe the usage of a generated editor for simple activity diagrams in the extended GMF version. A sequence of steps to create our sample activity diagram is shown in Fig. 3.

In Step 1, we start with an empty editor panel and select command *CreateStartGraph* from the palette. Immediately, the start activity diagram appears in the editor panel. Step 2 selects command *AddSimple* to add a simple activity node. This node is added after the start activity, because the negative application condition

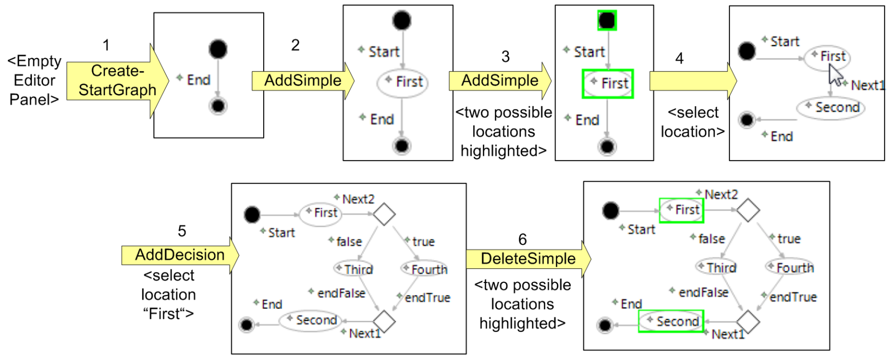

Fig. 3. Editing steps using complex editing operations in extended GMF

of the rule forbids to insert an activity node after a final node. Since we have only one non-final activity node, the location to apply this command is unique in the current situation. In Step 3, we select command *AddSimple* again, but this time it can be applied at two locations: the new activity node can be inserted either after the start activity, again, or after the new simple activity node named "First". Thus, instead of applying the rule, the editor now highlights the two possible locations. In Step 4, the "First" node is selected per mouse click, and the command is applied accordingly. Step 5 combines two atomic steps: command *InsDecision* is selected, and the activity nodes "First" and "Second" are clicked on to specify between which two activity nodes the complete decision structure is to be inserted. Afterwards, in Step 6, command *DeleteSimple* is selected in the palette. This leads to two activity nodes being highlighted, which may be deleted by the rule.

4 Outlook

Using pure GMF, a visual editor may be generated which offers basic editor commands for each model element, only. In this documentation, we described an extension of GMF for generating visual editors [5] which provides complex editing commands. For the generation of complex editor commands, an additional model is needed. We use EMF transformation rules to formulate commands based on the given domain model. To the best of our knowledge, no other meta CASE tool based on meta models offers the possibility to define complex editing commands.

Besides pure editing commands, also model optimizations such as model refactorings, may be realized by the presented approach. Moreover, model transformation rules can also be used to simulate behaviour models. Thus, this work can be considered as a starting point for the generation of powerful and flexible domain-specific visual editors in Eclipse.

References

1. Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., Weiss, E.: Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In: Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006. LNCS, Springer, 2006. `http://tfs.cs.tu-berlin.de/emftrans`
2. Eclipse Consortium, *Eclipse Graphical Editing Framework (GEF)*, 2007, available at `http://www.eclipse.org/gef`.
3. Eclipse Consortium, *Eclipse Graphical Modeling Framework (GMF)*, 2007, available at `http://www.eclipse.org/gmf`.
4. Eclipse Consortium, *Eclipse Modeling Framework (EMF)*, 2007, available at `http://www.eclipse.org/emf`.
5. Tiger Developer Group, *Tiger GMF Transformation Project*, 2007, available at `http://tfs.cs.tu-berlin.de/emftrans/gmftrans`.
6. Ehrig, K. and Ermel, C. and Hänsgen, S. and Taentzer, G., *Generation of Visual Editors as Eclipse-Plugins*. Automated Software Engineering'05, IEEE Computer Society, 2005, `http://tfs.cs.tu-berlin.de/tigerprj`.