eMoflon@TTC2013: Integrating Java Source Code with FlowGraph Models

Anthony Anjorin and Marius Lauder

Real-Time Systems Lab
Technische Universität Darmstadt (Germany)

This document describes very briefly all actions that are required to specify and apply a bidirectional model transformation based on Triple Graph Grammars between `Java source code` *and* `FlowGraph` *models. All information regarding details of the specification is provided in an additional paper, submitted to the 2013 Transformation Tool Contest (http://planet-sl.org/ttc2013/).*

For further questions please contact us via contact@moflon.org

Specifying and using a TGG can be separated into two distinct steps:

(i) Specify the model integration on the metamodel level
(ii) Actually apply the derived forward/backward model transformation to an existing input model in a round-trip manner

In the following, both steps are briefly described to demonstrate, from a user's point of view, how bidirectional model transformation with TGGs can be achieved with eMoflon.

This introduction is by no means a complete introduction to TGGs or our tool eMoflon. If you are interested in additional details regarding eMoflon and its capabilities regarding metamodeling and (uni-/bidirectional) model transformations we refer to our extensive tutorial: www.emoflon.org

(i) Specifying a TGG and deriving the unidirectional operational rules

1. Start Eclipse (the workspace is already preconfigured).
2. Explore the Eclipse workspace:
    a. Project `FlowGraphs` contains the metamodels and the TGG specification.
    b. Projects `flowgraph`, and `JavaToFlowGraph` are placeholders for the language definitions that will be generated from the specified metamodels and TGG.
    c. The closed project `ttc-2013-flowgraphs-case` contains all information that was provided by the originators of the case study.
3. Open the `FlowGraphs.eap` file by double-clicking it
    a. Enterprise Architect Lite (EA) opens and allows viewing all details of the metamodels and TGG specification in a read-only fashion.
    b. Explore the metamodels which define the relevant structures for the integration.

   c. Explore the integration rules by navigating to the `Rules` package in the `JavaToFlowGraph` folder. Here you will find all TGG rules that have been defined to integrate `Java source code`- and `FlowGraph` instances.

4. As eMoflon is a fully generative approach, you have to export all information from EA to the Eclipse workspace where code generation will be performed: Right-click on any of the elements and select `Extensions` → `MOFLON::Ecore Addin` → `Export All to Workspace`.
5. Return to Eclipse and optionally check that all `gen` folders are empty.
6. Refresh `FlowGraphs` (select `Refresh` from the context menu). The workspace should now start building automatically.
7. Hand-written Java code to implement specific details of the transformation (e.g., user defined constraints) has been placed in the injection folders of the projects `flowgraph` and `JavaToFlowGraph`. Such code is merged automatically with generated code during the code generation process and, therefore, enriches generated code with additional functionality without comprising a clear separation.

(ii) Integration user part for applying model integration rules:

1. Navigate to the `JavaToFlowGraph` project and inspect the existing exemplary input models in the `instances/in` folder.
2. These models are going to be used as input for forward and backward transformation.
3. Right-click on `src/org.moflon/tie/TGGMain.java` and choose `Run As` → `Java Application` (the corresponding Java source code for executing a model transformation with these input models was generated automatically).
4. The transformations are performed and additional models are persisted in the `instances/out` folder (results of the round-trip starting beginning and ending with Java source code that should be semantically equivalent). Further models, such as the correspondence models are persisted in the folders `instances/Test0.java` and `instances/Test1.java.`
5. To visualize the transformation results, we provide an `Integrator` which is capable of presenting integrated model triples visually:
   (a) Navigate to the `instances/Test0.java` or `../Test1.java` folder.
   (b) The previously applied forward (backward) transformations created appropriate models with a `FWD` (`BWD`) flag.
   (c) Right-click on one of the `*_corr_*.xmi` files and select `eMoflon` → `Start Integrator`.
   (d) A new window opens where the integration result is visible:
      a. The left-hand side denotes source domain model (`Java` source code AST).
      b. The top row represents the target domain model (`FlowGraph` instance).
      c. The connections in the center denote the correspondence domain model (traceability).
   (e) Drag and drop the appropriate `*_protocol_*.xmi` into this view.

(f) You can now navigate back and forth through the transformation history by using the `ALT + arrow keys`. Here you can follow the actual transformation process, i.e., see at which point which elements are selected for transformation (blue), which elements are pending due to missing context dependencies (yellow) and which elements are successfully processed (grey). In the opposite domain, newly created elements are marked green.