Domain-Level Debugging for Compiled DSLs with the GEMOC Studio (Tool Demo)

Erwan Bousse, Tanja Mayerhofer, Manuel Wimmer

Domain-Level Debugging for Compiled DSLs with the GEMOC Studio

(Tool Demo)

Erwan Bousse
TU Wien
Vienna, Austria
Email: erwan.bousse@tuwien.ac.at

Tanja Mayerhofer
TU Wien
Vienna, Austria
Email: mayerhofer@big.tuwien.ac.at

Manuel Wimmer
CDL-MINT, TU Wien
Vienna, Austria
Email: wimmer@big.tuwien.ac.at

***Abstract*—Executable Domain-Specific Languages (DSLs) are commonly defined with either operational semantics (*i.e.,* interpretation) or translational semantics (*i.e.,* compilation). An interpreted DSL relies on domain concepts to specify the possible execution states and steps of conforming models, which facilitates the observation and control of the execution using the very same domain concepts. In contrast, a compiled DSL relies on a transformation to an arbitrarily different executable target language, which creates a conceptual and technical gap between the considered domain and the target domain. In this tool demonstration paper, we present the implementation of our approach to supplement a compiled DSL with a *feedback manager*, which during execution translates execution steps and states of the target model back to the source domain. This enables the development and use of tools such as an omniscient debugger and a trace constructor for debugging compiled models. Our implementation was achieved for the GEMOC Studio, a language and modeling workbench that provides generic model debugging tools for interpreted DSLs. With our approach, these debugging tools can be also used for compiled DSLs. Our demonstration features the definition of a feedback manager for a subset of fUML that compiles to Petri nets.**

I. Introduction

A large amount of Domain-Specific Languages (DSLs) have been proposed to describe the dynamic aspects of systems. Defining the execution semantics of such DSLs opens the possibility to use *dynamic verification and validation* (V&V) techniques, such as interactive debugging or tracing. Two approaches are commonly used to define the semantics of an executable DSL: operational semantics (*i.e.,* interpretation) resulting in an *interpreted DSL*, and translational semantics (*i.e.,* compilation) resulting in a *compiled DSL*.

Dynamic V&V techniques rely on two key tasks: the observation of the progress of the execution (*e.g.,* which execution steps are occurring), and the control of the execution (*e.g.,* pausing and resuming). In the case of an interpreted DSL, the dynamic state of a model is defined along with the possible execution steps that modify such state over time. These definitions can directly rely on domain-specific concepts, opening the possibility to observe and control executions from a domain perspective (*e.g.,* visualizing the current token distribution in an activity diagram). However, in the case of a compiled DSL, a model is translated into a model conforming to another executable language. By default, observing the resulting execution yields information specific to the target language instead of the considered DSL, *i.e.,* there is no *feedback* at the domain level. For instance, when lower-level code or models are generated from a model, it is common to only rely on the debugger of the target language to debug the execution, but without the possibility to directly relate information back to the original model.

In this paper, we present the implementation of our approach to supplement a compiled DSL with a *feedback manager*. During the execution of a model, this component translates execution steps and states of the target model back to the source domain. Generic tools, such as an omniscient debugger (*i.e.,* a tool to explore past and future execution states) and a trace constructor, can thereby be used for debugging compiled models at the domain level. We realized this implementation for the GEMOC Studio, a language and modeling workbench. Our demonstration shows the definition of a feedback manager for a subset of fUML that compiles to Petri nets.

II. The GEMOC Studio

The GEMOC Studio (http://gemoc.org/studio) is a workbench atop the Eclipse Modeling Framework (EMF) including:

- The *GEMOC Language Workbench*, used by language designers to build and compose new executable DSLs.
- The *GEMOC Modeling Workbench*, used by domain designers to create, execute models conforming to DSLs.

The different concerns of an executable DSL, as defined with the tools of the language workbench, are deployed into the modeling workbench. This configures a generic execution framework that provides various generic runtime services, such as graphical animation, omniscient debugging, trace and event managers, timeline visualizations, etc. Originally, the GEMOC Studio focused on providing facilities to design and implement *interpreted* DSLs. In this work, we extend GEMOC Studio to also support the design and implementation of compiled DSLs, thereby enabling the application of existing runtime services and model execution tools for compiled models.

III. Compiled DSLs in the GEMOC Studio

We extended the GEMOC Studio with facilities to define and use compiled DSLs. The source code of the presented extension to the GEMOC Studio, along with the example, is

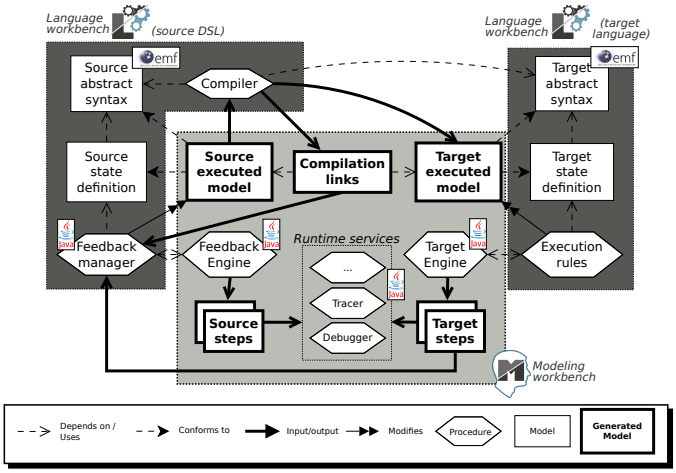

Fig. 1. Overview of the architecture.

currently available on Github[1] under the EPL 1.0 license, and consists in multiple Eclipse plugins written in Xtend and Java.

Figure 1 shows an overview of the approach. The source DSL is shown on the left, with translational semantics realized by a compiler. This compiler is a model transformation from the source abstract syntax to the target abstract syntax, defined with any model transformation language compatible with Ecore metamodels. The core part of our approach is the *feedback manager* of the source DSL, which translates target states and steps into source states and steps. It is defined by the language engineer using any Java-compatible language. The language designer must also define what is the *state* of a model conforming to the source DSL. Both source and target languages are defined in the GEMOC Language Workbench.

In the middle, the execution of the source model in the GEMOC Modeling Workbench is shown. First, the source model is compiled into a target model. Then, the execution of the target model is performed by an *execution engine* using execution semantics of the target language[2]. Using this mechanism, the feedback manager of the source DSL registers itself as a listener of the target engine. Then, the feedback manager is used by the generic *feedback engine* to translate at runtime the target states and steps into source states and steps. The feedback engine finally notifies its own set of listeners (*e.g.,* an interactive debugger or a tracer) about occurring source execution steps, *i.e.,* at the domain-level. Concretely, these listeners are oblivious of the underlying target model execution, and only perceive that the source model is actually being executed. They can therefore be used both with interpreted and compiled DSLs.

IV. Demo

The first part of the demonstration shows the point of view of the language engineer in the GEMOC Language Workbench while implementing an fUML activities DSL that compiles to Petri nets. Figure 2 is a screenshot of the expected workbench. At the top-left corner, the Ecore abstract syntaxes of both languages are shown. At the top-right corner, the Kermeta semantics of the target language is shown. At the bottom left corner, an excerpt of the compiler is shown. Although it is implemented in Xtend, any model transformation language can be used. At the bottom right corner, an excerpt of the feedback manager is shown, also written in Xtend.

The second part of the demonstration consists in showing the point of view of the modeler while executing an fUML activity diagram model in the debugging environment of the GEMOC Modeling Workbench. Figure 3 is a screenshot of the expected workbench. At the bottom left corner, the executed activity is shown, while the target Petri net obtained by compilation is shown to its right. This Petri net is executed by an interpreter, while the feedback manager of the activity diagram DSL continuously translates target steps and states back to the activity diagram model. Thanks to the feedback, the activity diagram is automatically animated during the execution of the Petri net (*i.e.,* the token flow is displayed).

The remaining panels are all dedicated to debugging, and indirectly rely on information provided by the feedback manager. At the top left, the stack panel displays all ongoing activity diagram execution steps. Thanks to the feedback manager, it shows that the activity fork node is currently offering a token, instead of showing that a transition of the Petri net is being fired. At the top right, the variable view shows the dynamic data contained in the activity diagram state updated by the feedback manager, *i.e.,* the tokens held by nodes and edges. At the bottom right, the execution trace being constructed is shown, which represents the activity diagram execution instead of the Petri net execution. This trace is used by the underlying omniscient debugger, which can be used to revisit past states of the activity diagram by double clicking on the states.

V. Conclusion

We have shown the implementation of our approach to provide domain-level debugging facilities for compiled DSLs in the GEMOC Studio. Future work includes the management of compilers written as code generators, and more generally facilitating the use of target languages not initially implemented with the GEMOC Language Workbench.

Acknowledgments

This work has been funded by: the Austrian Science Fund (FWF): P 28519-N31; the Austrian Agency for International Mobility and Cooperation in Education, Science and Research (OeAD) on behalf of the Federal Ministry for Science, Research and Economy (BMWFW) under the grand number FR 08/2017, by the French Ministries of Foreign Affairs and International Development (MAEDI), and the French Ministry of Education, Higher Education and Research (MENESR); and the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development.

[1] https://github.com/ebousse/gemoc-compilation-engine

[2] If the target language is interpreted, the execution engine uses the operational semantics of the DSL. If the target language is compiled, then the engine is a feedback engine itself and behaves as explained in this section.

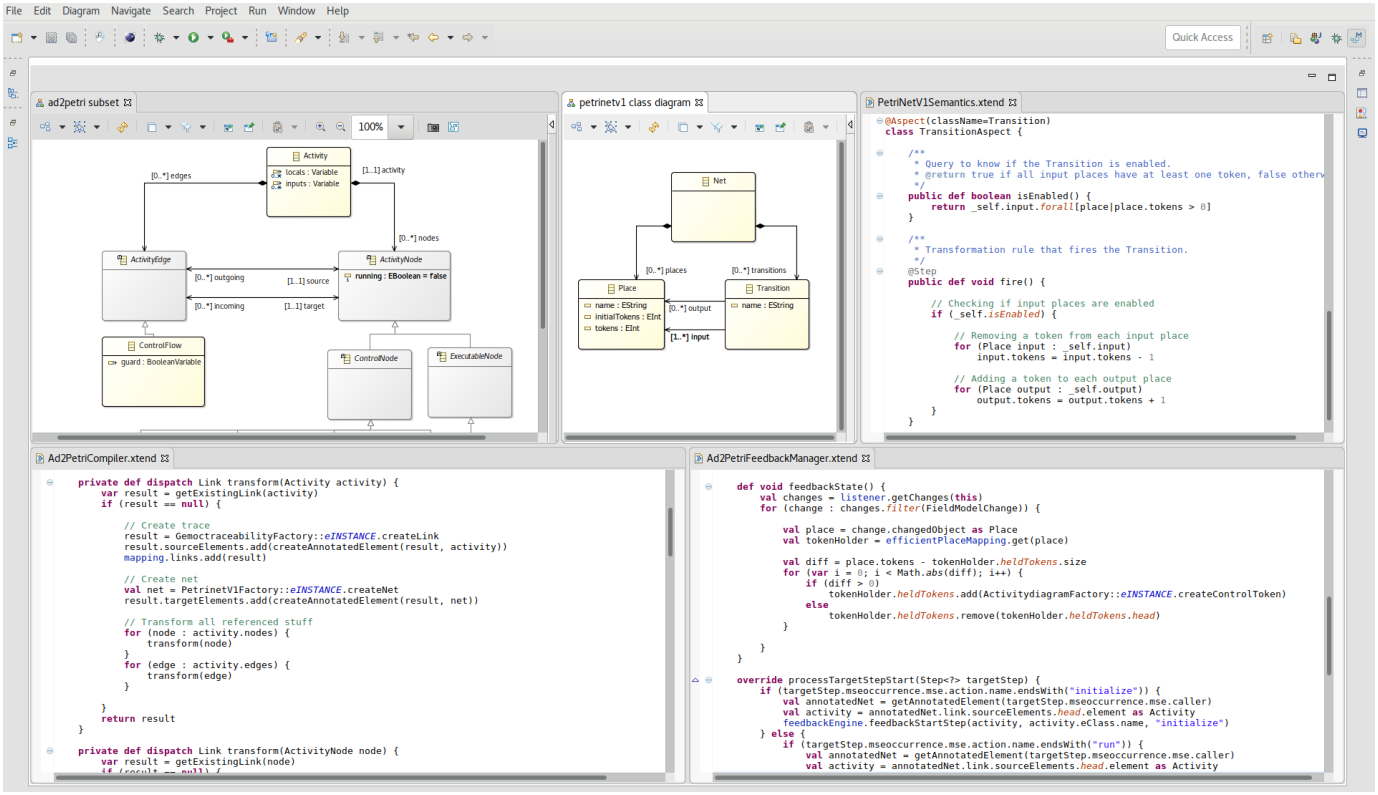

Fig. 2. Screenshot of the GEMOC Language workbench, while implementing a compiled activity diagram DSL. The abstract syntaxes of the source language (activity diagram, top left) and target language (Petri nets, top middle) are shown, along with the semantics of the target language (top right), the compiler (bottom left) and the feedback manager (bottom right).

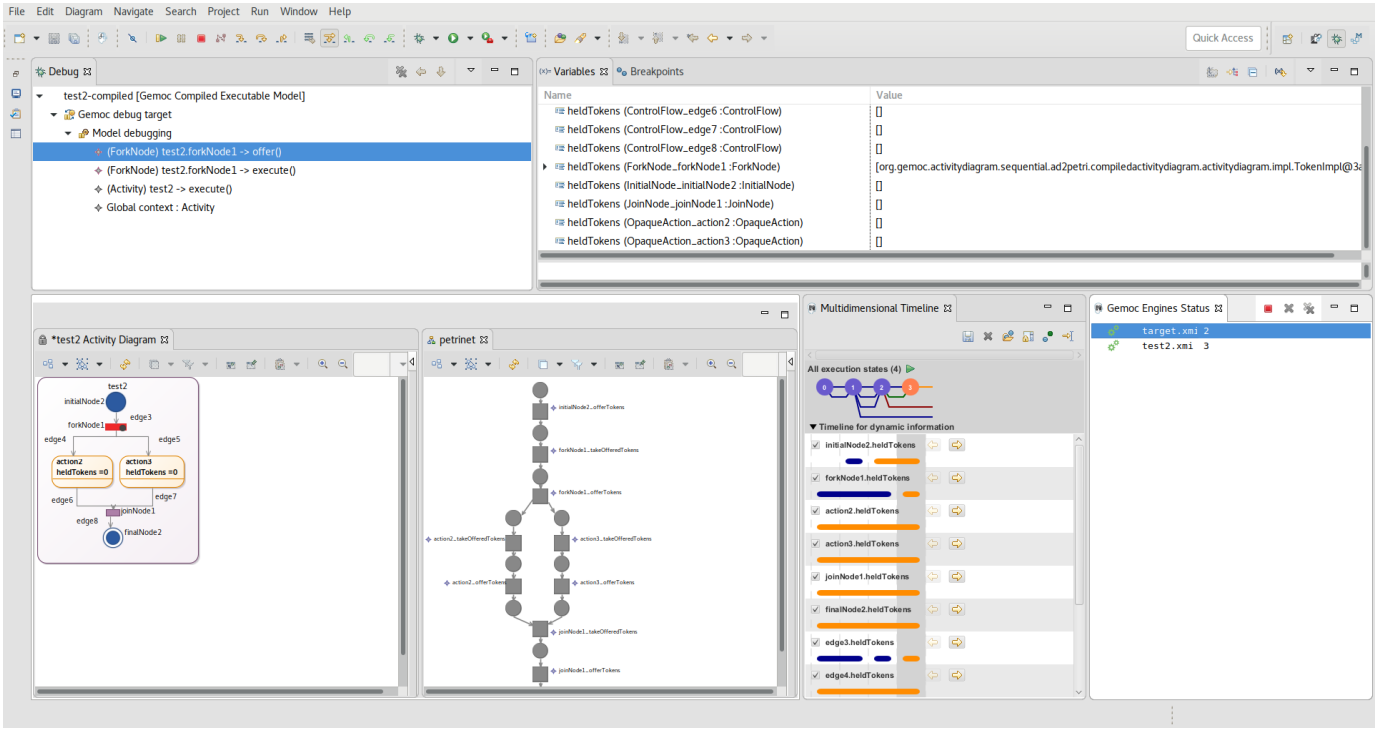

Fig. 3. Screenshot of the GEMOC Modeling Workbench, during an omniscient debugging session of an activity diagram (bottom left) compiled to a Petri net (bottom center). While the underlying Petri net is being executed, feedback is provided by animating the activity diagram (bottom left), showing the stack of domain-level steps (top left), current domain-level dynamic data (top right), and a domain-level execution trace (bottom right).