Frameworks to Create Language Definitions and Tools on Top of the Eclipse Modelling Project

Hajo Eichler, Daniel A. Sadilek, Markus Scheidgen, Michael Soden, Guido Wachsmuth, Stephan Weißleder

Introduction

In today’s software engineering world, languages are only as good as their supporting tools, and engineering processes are only as good as the languages used in them. New engineering paradigms, domain specific engineering, and the general call for more efficient software engineering cause high demand for new programming and modelling languages including high quality language tools.

The eclipse platform, especially the Eclipse Modelling Project with EMF, QVT, GMF, EMFT, and MDT provides the ideal backbone for the generative development of languages and language tools.

Over the last years, we developed a series of experimental frameworks based on EMF that enhance the possibilities to describe language in more detail and derive even better tools from these descriptions. These frameworks tackle all language aspects, from notations to semantics: we can write, execute, analyse, debug, refactor, test, and compile models for different target platforms. This poster shows a brief display of our work.

References


“Eclipse Modelling Project in combination with our frameworks provides powerful facilities to create languages and their tools from head to toe”

Smimco – Simulating and Compiling Models for Specific Target Platforms

Smimco allows to describe a modelling language’s execution semantics. It is based on bridging EMF and the Scheme programming language. Smimco provides three options to define the execution semantics:

(1) a transformation of an EMF-based model to Scheme code
(2) an interpreter that works on a Scheme representation of the EMF-based model
(3) an interpreter that is implemented on the language’s metamodel

No matter which option the user chooses, Smimco allows to use the execution semantics description in two ways:

(1) a model can be simulated in the development environment: A discrete event simulation kernel allows to simulate multiple runs of a model with time dependencies, e.g. to simulate a distributed system.
(2) a model can be compiled for a target platform: Each option to define the execution semantics of a language results in a Scheme representation of models. After providing bindings for the target platform, e.g. access to special hardware, this representation can be compiled with standard DSL editors:
- text editors
- model editors that look like text editors
- text editors embedded in your graphical DSL editors created with GMF

TEF – Textual Model Notations and Editors

The Textual Editing Framework allows to create full fledged textual editors with all options to define the abstract syntax of a language, changing over time. With the help of QVT relations, we map between different levels of abstraction, e.g. to import parsed source code and create executable models.

EProvide – Prototyping Visual Interpreters and Debuggers

With EProvide, a user can prototype visual interpreters and debuggers for a new modelling language. For this, he defines the structural operational semantics as a QVT transformation, extends the metamodel with concepts to represent simulation runs. The overall framework provides a multi layer view with an explicit instantiation concept where runtime models are regarded as instances of the abstract syntax, changing over time.

With the help of QVT relations, we map between different levels of abstraction, e.g. to import parsed source code and create executable models.

M3 Actions – Metamodels with Behaviour

M3 Actions is a MOF extension to specify operational semantics of models. Main goal is dynamic model analysis of executable specifications through simulation and test integration.

Build on top of EMF/GMF, OCL and QVT the framework consists of a visualeditor to define metamodels, runtime models, and their behaviour: An interpreter and debugger supports the execution and testing of these models as well as recording of simulation runs. The overall framework provides a multi layer view with an explicit instantiation concept where runtime models are regarded as instances of the abstract syntax, changing over time.

ECoral – Metamodel Adaptation with Co-Adaptation of Models

Usually, you adapt your Ecore metamodels manually in an ad-hoc fashion. With ECoral, you specify transformational adaptation in scripts, using a library of well-defined adaptation steps.

Metamodel adaptation usually requires co-adaptation of models and model transformations. Simply apply them, and your models and model transformations are compliant to your new metamodel.

MMUnit – Automated Metamodel Testing

The abstract syntax of a language can be defined with Metamodels. Metamodels contain errors. Nevertheless, they are not tested systematically, which leads to late error detection and higher error correction effort. By testing the Metamodel of a language early, this effort can be saved. MMUnit supports early Metamodel testing with JUnit. It is based on a test Metamodel, whose instances describe positive and negative example models of the Metamodel under test.