Tool Demo: The MPS Language Workbench

Markus Voelter
independent/itemis
voelter@itemis.de

Vaclav Pech
JetBrains
vaclav.pech@jetbrains.com

***Abstract*—JetBrains MPS is a comprehensive environment for language engineering. New languages can be defined as standalone languages or as modular extensions of existing languages. Since MPS is a projectional editor, syntactic forms other than text are possible, including tables or mathematical symbols. This demo will show MPS based on mbeddr C, a novel approach for embedded software development that makes use of incremental language extension on the basis of C.**

***Keywords*-language engineering, language extension, language composition**

I. Introduction and Motivation

Finding and working with the right abstractions for describing a problem or its solution is one of the central pillars of software engineering. Once the right abstractions have been found, programs can be expressed more concisely, are easier to understand, the description can be analysed and other artifacts can be synthesized. A language is the purest form of abstraction, adding a suitable concrete syntax to work with the abstractions effectively. An IDE that supports syntax coloring, code completion, error annotation and refactoring makes working with the language and its abstractions even more productive.

Many software systems are composed from several concerns, each of them expressed with its own set of abstraction, possibly by different people, some of them may not even be programmers. To describe a real-world system well, several languages are necessary.

Looking at software engineering in this light, an environment is needed in which language creation is easy, languages can be composed and the concrete syntax is very flexible to encompass the needs of all the stakeholders involved in the software system. The JetBrains MPS open source language workbench is such as system.

II. About MPS

MPS[1] is an open source language workbench, developed by JetBrains and licensed under Apache 2.0. MPS is a projectional editor. This means that a wide variety of syntactic forms are supported including textual, symbolic (e.g. mathematical), tabular and, in the upcoming version 3.0, graphical (see the decision table in Fig. 1). Even for textual notations, no grammar or parser is used, which means that language composition cannot lead to ambiguous grammars — ambiguities have to be resolved by the user when entering a program. IDE support is provided for all notations. While projectional (aka structural) editors have had a bad reputation traditionally, MPS has managed to make the editing experience very much like traditional text editing.

III. About mbeddr C

mbeddr C[2] is an extensible version of the C programming language implemented in MPS. It is open source under the Eclipse Public License and can be found at `http://mbeddr.com`. Development teams can build their own domain-specific extensions of C to increase productivity and reliability in embedded software development. mbeddr C comes with a couple of reusable extensions such as interfaces, components, unit test support and state machines. It also integrates model checking and SAT solving to support formal verification of software systems. By extending programming languages with suitable abstractions, verification becomes significantly simpler.

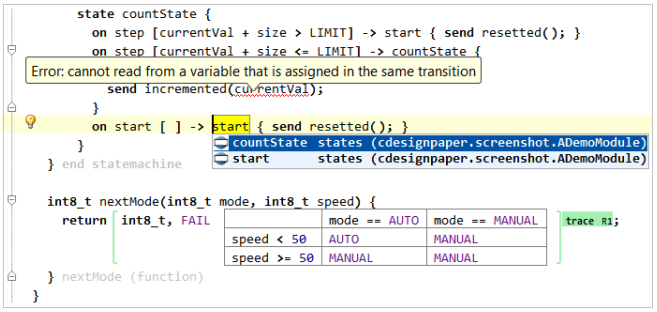

Figure 1. An example of a program written with mbeddr C in MPS. It mixed C functions, state machines and decision tables, each of them defined as a separate language. The screenshot also shows requirements traces (green annotations), which can be attached to any program element without the definition of this element being aware of the annotation.

IV. Contents of the Tool Demo

Introduction We will introduce the need for language engineering in the same way as in the introduction above.

[1]http://jetbrains.com/mps

[2]http://mbeddr.com

Example Use Case We will then show an example from the mbeddr.com extensible C language. The project develops modular extensions of C to make embedded software development more productive. The project demonstrates an intriguing use case fo language extension and showcases syntactic forms other than text, such as the decision table shown in.

Creating Extensions The main part of the demo will consist of a demo of how to build a language extension. Depending on the time we have in the tool demo, we will build the Hello World of language extensions, the `unless` statement. It is basically a negated `if`. This extension involves the following steps:

- Defining a new language that extends C so we can "plug into" C's statements
- Defining a new concept `UnlessStatement` that extends C's `Statement` and has an `Expression` and a `StatementList` as children
- Creating an editor that projects the `UnlessStatement` in the same way as the `if`, with the keyword `unless` instead
- Defining a type system equation that ensures the expression in the `UnlessStatement` is Boolean
- Implementing a generator that maps the `unless` back to an `if` with a negated condition, so it can be compiled with a regular C compiler
- And finally, a quick fix that allows the in place transformation of existing `if` statements to an `unless` (if they don't have `else if` or `else` clauses).

This language extension shows most of the essential ingredients to language definition in MPS.

V. What the audience will take away

The audience will understand the benefits of language extension. They will realize that, with the right tools, the efforts for defining extensions is limited, and can be done as part of real-world software development projects.

VI. Further Reading

More details about the mbeddr approach and the capabilities of MPS can be found in the technical report at `http://bit.ly/wIYERh`.