Tool Demo: The MPS Language Workbench

Markus Voelter
independent/itemis
voelter@itemis.de

Vaclav Pech
JetBrains
vaclav.pech@jetbrains.com

***Abstract*—The 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. In this session, we will TODO**

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

I. Introduction and Motivation

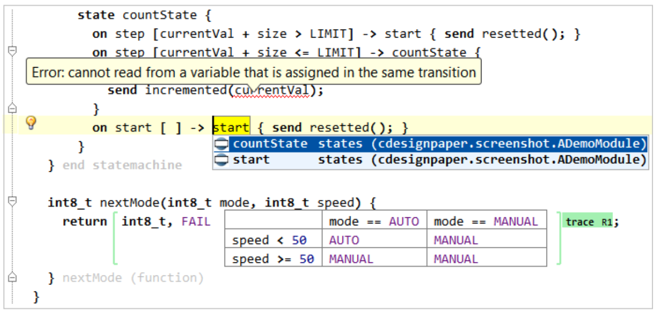
Finding and working with the right abstractions for describing a problem or its solution is one of the central pillars of software engineering. Once the right abstractions have been found, things can be expressed more concisely, are easier to understand, the description can be analysed and other artifacts 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. So, to describe a real-world system, ideally 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. How MPS Works

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. 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.



```
      state countState {
        on step [currentVal + size > LIMIT] -> start { send resetted(); }
        on step [currentVal + size <= LIMIT] -> countState {
Error: cannot read from a variable that is assigned in the same transition
          send incremented(currentVal);
        }
        on start [ ] -> start { send resetted(); }
      }                 countState  states (cdesignpaper.screenshot.ADemoModule)
    } end statemachine  start       states (cdesignpaper.screenshot.ADemoModule)

    int8_t nextMode(int8_t mode, int8_t speed) {
      return [ int8_t, FAIL |             | mode == AUTO | mode == MANUAL |
                            | speed < 50  | AUTO         | MANUAL         |
                            | speed >= 50 | MANUAL       | MANUAL         | ] trace R1;
    } nextMode (function)
  }
```

Figure 1. An example of a program written with 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.

III. Contents of the Tool Demo

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

Example Use Case We will then show an example from the mbeddr.com extensible C language [1]. 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 Fig. 1.

Creating Extensions The main part of the demo will consist of a demo of how to build a lanugage extension,

IV. What the Audience will take away

Acknowledgment

The authors would like to thank... more thanks here

References

[1] mbeddr - embedded development using c language extensions, http://mbeddr.com.