Agile Development with Domain Specific Languages

Bernhard Rumpe, Martin Schindler, Steven Völkel, and Ingo Weisemöller

Software Engineering
RWTH Aachen University, Germany
`http://www.se-rwth.de/`

1 Introduction

An increasing number of software development projects uses domain specific languages (DSLs) at least at one stage. Such languages allow domain experts to take part in the product development, and they can often contribute to improved efficiency. As a drawback, the development of a DSL is a complex and error-prone software development process itself, which causes additional efforts and costs. Moreover, the actual software product and the DSL are often developed concurrently, and the requirements for the DSL may change according to the needs of developers of the actual product. Therefore, we have to address two interdependent development processes: the product development process, in which we may need to react on requirement changes by the customer quickly, and the language development process, in which we want to define an adaptable and extensible DSL.

In this tutorial, we sketch preliminary considerations about the use of DSLs, important methods and techniques that are crucial for defining the language, and basic technologies for code generation. In our tutorial, we also introduce concepts for the modular definition of DSLs, quality assurance, and the integration of a DSL into software development processes. Both our tutorial and this summary build on our tutorial *Generative Software Development* presented at the ICSE 2010, respectively on the corresponding summary [4].

2 Usage of DSLs in Software Development Processes

There is already a considerable number of successfully applied domain specific languages. In the requirements and analysis phase, requirements specification languages that are close to natural languages have been introduced. Architectural description languages and the UML play an important role in system design. Matlab/Simulink is a wide spread language for the implementation of electronic control units in the automotive industry.

The development of a domain specific language and the corresponding tools is a software development process itself, which may be expensive and error-prone. Therefore, the introduction of a DSL is particularly useful in the development of large and complex products [1]. In smaller development processes, the improvements in terms of efficiency and software quality may not be sufficient to compensate the initial costs that are caused by the DSL development. If in contrast the complexity, size and lifetime of the software product are sufficiently large, the development of high-quality languages and language instances can substantially contribute to a more efficient and valuable software system.

3 Development of Domain Specific Languages

The development of precise DSLs and accompanying tools like MontiCore [3] contain concepts of metamodels or grammars (syntax), context conditions (static analysis and quality assurance) as well as possibilities to define the semantics of a language. Instances of most DSLs can be mapped to models in different languages or executable programs by model transformations and code generators. The growing number and complexity of DSLs is addressed by concepts for the modular and compositional development of languages and their tools.

As a first step the language has to be defined precisely. This includes a description of the valid words of the language, which is determined by its syntax and by context conditions. These are often described by means of context free grammars, attribute grammars, symbol tables and constraints.

The language definition also includes a description of the semantics of the language [2]. This is often implemented by means of transformations, with code generators as an outstandingly important special case. In the case of executable models, the target language is often a general purpose language such as Java or C++, and the runtime semantics of the source model are the runtime semantics of the generated code. Most model-to-text-transformations are implemented by means of template languages such as Freemarker. Transformations can be executed locally on the developer's machine, or remote as a transformation service, where the latter reduces the technical efforts required for using a DSL.

In addition to the language definitions, developers need tools to describe and transform models in the DSL, and the process must be adopted to the usage of the new language. Moreover, measures for quality assurance of documents in the language are required. Once these steps in language and tool development have been completed, the DSL is ready to be used in other software development processes.

References

1. Deursen, A., Klint, P.: Little Languages: Little Maintenance? Journal of Software Maintenance: Research and Practice 10, 75–92 (1998)
2. Harel, D., Rumpe, B.: Meaningful Modeling: What's the Semantics of "Semantics"? Computer 37(10), 64–72 (2004)
3. MontiCore Website: `http://www.monticore.de/`
4. Rumpe, B., Schindler, M., Völkel, S., Weisemöller, I.: Generative software development. In: Proceedings of the 32nd International Conference on Software Engineering (ICSE 2010). vol. 2, pp. 473–474. ACM (May 2010), tutorial summary