Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

COMPARISON OF SLX AND MODEL-DRIVEN LANGUAGE DEVELOPMENT FOR CREATING DOMAIN-SPECIFIC SIMULATION LANGUAGES

Andreas Blunk
Joachim Fischer

Humboldt-Universität zu Berlin
Rudower Chaussee 25
D-12489 Berlin, GERMANY

ABSTRACT

Many approaches and tools exist for developing domain-specific languages (DSLs), each promising fast and cheap development, including language-specific tool support. In this paper, we compare two approaches for developing executable DSLs. The first one is SLX, an extendable language from the simulation community, based on a rich semantic foundation of core simulation constructs. The second one is a realization of a model-driven approach to language development based on several Eclipse Modeling Tools. It is centered around a metamodel, that defines the abstract syntax of a DSL, enriched by descriptions of possibly multiple different notations. We describe and compare the two approaches with respect to syntax description, execution semantics description, and automatic tool support. We then use this comparison to give some thought about a new approach that combines them.

1 INTRODUCTION

Domain-specific languages (DSLs) offer description means tailored to a specific application domain. They allow to describe certain kinds of problems in a more concise and readable way than with a general-purpose language (GPL) (Marjan Mernik 2005). One kind of DSL are external DSLs. They are of special interest to us because models expressed in them are easier to create and easier to understand. Their main obstacle is that assistant software tools, like an editor, and an execution engine usually have to be created by hand, which can be too expensive for small projects.

In this presentation we focus on the development of external simulation DSLs. We investigate two approaches and compare them to each other. The first approach is an extendable simulation language called SLX (Henriksen 2000). The second one is a realization of a model-driven approach to language development (MDLD) based on several Eclipse Modeling Tools. We illustrate their development capabilities by defining a simple state machine language (Harel 1987) with both approaches.

SLX is a simple process-oriented programming language with built-in simulation constructs. It allows to define a DSL as a set of extensions to a core language. Models can be expressed as a mix of core and extension constructs. The new constructs are given a distinct syntax and their semantics are defined as a mapping to the core language.

In MDLD, DSL development is centered around a metamodel-based abstract syntax definition (OMG 2011). Other language aspects, e.g. concrete syntax and execution semantics, are defined in special description languages by referring to the metamodel. In addition, there exist mechanisms for automatically generating DSL tools. The concrete MDLD tools used in this comparison are the Eclipse Modeling Framework (EMF) (EMF 2012) for the definition of a metamodel, Xtext (Xtext 2012) for defining a textual notation, and Acceleo (Acceleo 2012) for the description of execution semantics in the form of a model to text transformation.

978-1-4673-4781-5/12/$31.00 ©2012 IEEE

2 COMPARISON

In SLX, DSL development is limited to adding new forms of statements and expressions. The set of languages that can be described are regular languages. This allows to describe only simple DSLs. Nevertheless, DSL description means are also simple and easy to learn. On the contrary, there exists the EBNF-like notation description language Xtext, which allows to define DSLs of the larger set of context free languages. In addition, the metamodel foundation allows to define language abstractions, which can be used for several similar DSLs.

The representation of DSL models is only textual in SLX, i.e. there is no interface for accessing the structural parts of DSL models. In MDLD, DSL models can be represented in many different ways, e.g. as XML for model serialization, in a GPL for tool implementation, and by a custom syntax for easy model creation.

For the description of an execution semantics, MDLD can be used in a much more flexible way than SLX. The parts of a DSL model can be accessed in a language, which is specially suited for this purpose. In addition, different ways of expressing the semantics exist. One of them is describing semantics as a model to text transformation, e.g. by using Acceleo.

SLX has an advantage when a DSL makes use of general constructs. With SLX, the already existing constructs of the core language can be used jointly with DSL constructs. In MDLD, the addition of GPL constructs is more complicated. In Xtext, an expression language is already defined and can be integrated into a DSL. However, in this case execution semantics have to expressed in the language Java.

Besides its simple extension mechanism, the execution speed of SLX models is very fast. Models are translated to runtime efficient machine level assembler code. In comparison to a C++-based simulation library, SLX executes up to a hundred times faster.

3 CONCLUSION

The comparison shows that SLX has a simple but easy to use extension mechanism for defining DSLs. Such DSLs can instantly benefit from programming language concepts provided by the SLX core language. In addition, SLX programs are executed very fast. MDLD has its advantages when it comes to model representation and additional tool support, which is supported by various open-source tools in the Eclipse community. Language tools can be derived automatically and they offer language-specific features. But execution speed can be problematic in this approach.

A combination of the DSL descriptions means of SLX and MDLD seems to be promising. Our future project will be to create such a combined approach. We want to add an extension mechanism based on the one in SLX but with the capability of extending the core language in a more flexible way. Core language concepts should be easily usable in a language extension and they should be extendable in themselves to a certain degree.

REFERENCES

Acceleo 2012. “Acceleo - transforming models into code”. http://www.eclipse.org/acceleo/.

EMF 2012. “Eclipse Modeling Framework (EMF)”. http://www.eclipse.org/modeling/emf.

Fowler, M. 2011. *Domain-Specific Languages*. Addison Wesley.

Harel, D. 1987, June. “Statechars: A Visual Formalism for Complex Systems”. *Science of Computer Programming* 8 (3): 231–274.

Henriksen, J. O. 2000. “SLX: The X is for Extensibility”. *Proceedings of the 2000 Winter Simulation Conference*.

Marjan Mernik, Jan Heering, A. M. S. 2005. “When and How to Develop Domain-Specific Languages”. *ACM Computing Surveys, Vol. 37, No. 4, pp. 316-344*.

OMG 2011. “Meta Object Facility (MOF) Core Version 2.4.1”. http://www.omg.org/spec/MOF/.

Xtext 2012. “Xtext Version 2.1 Documentation”. http://www.eclipse.org/Xtext/documentation/.