Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

PRACTICAL EXPRESSIVENESS OF INTERNAL AND EXTERNAL DOMAIN-SPECIFIC MODELING LANGUAGES

Tom Warnke
Adelinde M. Uhrmacher

University of Rostock
Albert-Einstein-Str. 22
D-18059 Rostock, GERMANY

ABSTRACT

During the long history of modeling and simulation, many answers have been given to the question of how to specify simulation models. Many of these approaches can be perceived as domain-specific modeling languages offering a syntax and a semantics. However, the individual languages are often vastly different. A central distinguishing aspect is the classification as external or internal domain-specific language. External and internal domain-specific languages are characterized by specific trade-offs regarding syntactical flexibility, computational efficiency, and amount of implementation work. We present a case study of alternative approaches to implement domain-specific languages for a small modeling problem in supply chain management. We illustrate the influence of using an external or internal language on different aspects of language performance, in particular the practical expressiveness, which we identify as one of the central properties of modeling languages.

1 DOMAIN-SPECIFIC MODELING LANGUAGES

One of the main contributions of the scientific discipline of modeling and simulation is the separation of model and simulation concerns. The *model* abstracts a system of interest, approximating it with a set of assumptions. The *simulation* produces data from the model by stimulating its inputs and recording its outputs (Cellier 1991). By adhering to this separation of concerns, many challenges that arise when dealing with simulation models can be handled much more efficiently. However, despite being separated from the execution algorithms, the model description must be clear and unambiguous. One way to describe simulation models is to use a *domain-specific modeling language* (DSML).

Domain-specific languages (DSLs) have been successfully applied in many fields in computer science (for an overview see Van Deursen, Klint, and Visser (2000)). The defining property of DSLs compared to General Purpose Languages (GPLs) is their better expressiveness, but limited application domain. Frequently cited examples for DSLs include SQL, makefiles, or LaTeX, which exemplifies that the idea is applicable in different domains. In general, the advantages of DSLs are widely acknowledged. However, it is equally acknowledged that designing a good DSL is hard (Mernik, Heering, and Sloane 2005).

One of the reasons that designing a good DSL is hard is that expressiveness of a (programming) language is not clearly defined. While it is often interpreted in terms of theoretical expressiveness, for most DSLs the *practical expressiveness* is more important. Rather than evaluating programming languages in terms of the programs that can be expressed, practical expressiveness refers to aspects such as the conciseness, readability, or the abstractions of programs written in a given language (Felleisen 1991). A precise definition of practical expressiveness is still elusive; however, the underlying idea was expressed in quotes such as "Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy" (Perlis 1982) and "Simple things should be simple, complex things should be possible" (Alan Kay in Leuf and Cunningham (2001)).

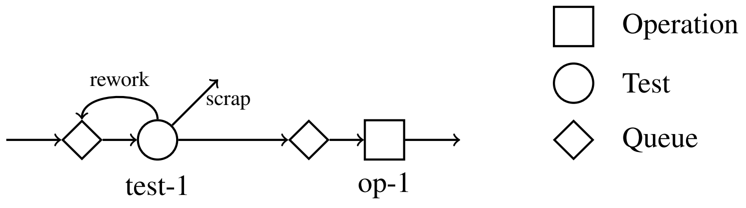

Figure 1: Schematic view of a simple supply chain model, in which items pass through a sequence of operations and tests (Persson and Olhager 2002).

Two general approaches to achieve practical expressiveness in DSLs can be distinguished. First, *external* DSLs define arbitrary syntax and semantics, and thus enjoy the maximal freedom in designing the language. The language designer can adapt the language to the needs of the application domain and directly provide the desired ease of use. However, the designer also has to provide a tool chain (editor, lexer, parser) to make the language usable. Second, *internal* (or embedded) DSLs build upon an existing GPL by defining an expressive application programming interface (API). Thus, when writing in the DSL, the language user actually writes a program in the host language and can exploit available tools, language features, and libraries. These can provide the language with practical expressiveness as well, by allowing to delegate aspects of the model description to the underlying host language. The drawback of internal DSLs is that the language design is constrained by the syntax and semantics of the host language.

Both approaches have been applied to DSLs for modeling, or DSMLs. With a DSML, the model to simulate is defined textually and parsed or compiled to a computational model, which can then be input to a simulation algorithm. This way, the separation of model and simulation is supported.

2 CASE STUDY

To illustrate the differences in implementing different approaches to DSMLs, we showcase a small case study where we tackle a simulation problem with different DSML implementations. As an example use case for our DSMLs we chose the modeling of simple supply chains based on the work by Persson and Olhager (2002). Our DSMLs are able to capture simple variants of supply chain models such as the one represented graphically in Figure 1. Based on our DSML implementations, we offer some insights on the effect of different implementation strategies on the resulting languages. Specifically, we focus on the practical expressiveness of the languages, one of the important properties in practice.

REFERENCES

Cellier, F. E. 1991. *Continuous System Modeling*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Felleisen, M. 1991. “On the expressive power of programming languages”. *Science of Computer Programming* 17 (1): 35 – 75.

Leuf, B., and W. Cunningham. 2001. *The Wiki Way: Quick Collaboration on the Web*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Mernik, M., J. Heering, and A. M. Sloane. 2005. “When and How to Develop Domain-specific Languages”. *ACM Comput. Surv.* 37 (4): 316–344.

Perlis, A. J. 1982. “Epigrams on Programming”. *ACM SIGPLAN Notices* 17 (9): 7–13.

Persson, F., and J. Olhager. 2002. “Performance simulation of supply chain designs”. *International Journal of Production Economics* 77 (3): 231 – 245.

Van Deursen, A., P. Klint, and J. Visser. 2000. “Domain-specific languages: An annotated bibliography.”. *Sigplan Notices* 35 (6): 26–36.