Chapter 12
Demystifying Domain Specific Languages

Abdelilah Kahlaoui
École de Technologie Supérieure, Canada

Alain Abran
École de Technologie Supérieure, Canada

ABSTRACT

Domain Specific Languages (DSLs) provide interesting characteristics that align well with the goals and mission of model-driven software engineering. However, there are still some issues that hamper widespread adoption. In this chapter, the authors discuss two of these issues. The first relates to the vagueness of the term DSL, which they address by studying the individual terms: domain, specificity, and language. The second is related to the difficulty of developing DSLs, which they address with a view to making DSL development more accessible via processes, standards, and tools.

INTRODUCTION

The concept of Domain Specific Languages (DSLs) is not new, and the advantages to using them have long been highlighted in the literature:

We must develop languages that the scientist, the architect, the teacher, and the layman can use without being computer experts. The language for each user must be as natural as possible to her/him. The statistician must talk to his terminal in the language of statistics. The civil engineer must use the language of civil engineering. When a man learns his profession he must learn the problem-oriented languages to go with that profession (Martin, 1967, p. 89).

We must constantly turn to new languages in order to express our ideas more effectively. Establishing new languages is a powerful strategy for controlling complexity in engineering design; we can often enhance our ability to deal with a complex problem by adopting a new language that enables us to describe (and hence to think about) the problem in a different way, using primitives, means of combination, and means of abstraction that are particularly well suited to the problem at hand (Abelson, Sussman, & Sussman, 1996, pp. 359-360).

The use of DSLs is not limited to information technology, as they are used in many other areas as well, such as finance (Arnold, Van Deursen, &

DOI: 10.4018/978-1-4666-6042-7.ch012

Res, 1995), chemistry (Murray-Rust, 1997), biology (Hucka et al., 2003), music (Boulanger, 2000), for example. Among the DSLs most commonly used in computer science today are the Structured Query Language (SQL) (Chamberlin & Boyce, 1974; ISO, 2008), the regular expression language for manipulating strings (Friedl, 2006), Microsoft Office Excel (Microsoft, 2011), and the eXtensible Markup Language (XML) (ISO, 1986).

A DSL is usually designed to solve a specific class of problems in a particular domain. The focus of DSLs on a particular domain facilitates the creation of languages that best represent the domain concepts. This convergence between problem domain and solution domain has many benefits, in terms of the expressiveness and precision (semantics) of the DSL. In addition, DSLs have also shown good potential in terms of productivity, reusability, and reliability (Kelly & Tolvanen, 2008; Kleppe, 2008).

In this chapter, we discuss the concept of DSLs (section 1), and present the types of DSLs (section 2) and the tools used to develop them (section 3). Section 4 presents some of the standards that can be used as a foundation for developing DSLs. Section 5 describes the development process itself. Finally, we summarize this work in section 6.

BACKGROUND

As mentioned earlier, DSLs have been a part of the computing world for decades. SQL and the UNIX languages *awk* and *make* are a few examples. These languages were developed by specialists with solid language development skills and good knowledge of the DSL domain. However, the emergence of the model driven approach and the need for languages capable of producing precise models that can be processed by machines, has opened up new horizons for DSLs. In the future, these languages could play a central role in the software development cycle, which would take DSLs from the arena of specialists to that of software developers.

However, software developers generally do not have the skills required to develop DSLs, and so an effort must be made to make DSL development more accessible. Only two areas have been addressed up to now to achieve this: DSL tooling, and DSL development. On the one hand, tooling has been driven by the likes of IBM, Microsoft, and Metacase (see section 3 for further details about DSL tools). These companies offer tools designed to support most of the activities of the DSL development cycle. Unfortunately, these tools remain immature. On the other hand, there has been significant progress on individual aspects of DSL development. For example, (Mernik, Heering, & Sloane, 2005) have identified a set of patterns for the decision, analysis, design, and implementation phases of DSL development; (Tolvanen, 2006) provides guidelines and steps on how to create a DSL; (Deursen, Klint, & Visser, 2000) discuss DSL design methodology and provide a list of related publications, and (Thibault, Marlet, & Consel, 1999) propose a framework for designing and implementing DSLs.

Although this work helps demystify DSL development, it does not describe a well defined DSL development process that covers all its major aspects, namely: what process to follow, what products to use, what tools are required, and who is involved. So far, the research effort has mainly focused on DSL development phases and activities. For (Deursen, et al., 2000), a DSL development process comprises three phases: analysis, implementation, and use, while (Mernik, et al., 2005) identify five phases: decision, analysis, design, implementation, and deployment.

To improve the DSL development experience, three areas need to be considered:

1. Processes to provide a disciplined approach to DSL development;
2. Tools to support language development and maintenance; and