Domain-Specific Languages as Key Tools for ULSSIS Engineering

Jan Heering
CWI
Kruislaan 413, 1098 SJ Amsterdam
The Netherlands
Jan.Heering@cwi.nl

Marjan Mernik
University of Maribor
Smetanova 17, 2000 Maribor
Slovenia
marjan.mernik@uni-mb.si

ABSTRACT

We briefly discuss the potential of domain-specific languages and domain-specific modeling languages for ULSSIS engineering, some of the scaling challenges involved, and the possibilities for raising expressiveness beyond current levels.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*Languages*; D.3.2 [**Programming Languages**]: Language Classifications—*Specialized Application Languages*

General Terms

Design, Languages

Keywords

Domain-specific languages, ultra-large-scale systems

1. THE WEB AS DSL BREEDING GROUND

Domain-specific languages (DSLs) are languages tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general-purpose languages in their domain of application, with an attendant reduction in software complexity. They variously act as enablers of the use and reuse of domain knowledge, as tools for large-scale software development and software generation, and as tools for end-user development and human-computer interaction [2]. Contrary to the definition given in [3, Glossary], we emphasize that DSLs need not be programming languages in any conventional sense. In fact, one of their strengths is that they transcend the boundaries of programming proper. For instance, as domain-specific modeling languages (DSMLs) they are gaining a foothold in model-driven engineering (MDE) [1]. Another example is the well-known syntactic metalanguage EBNF, which is a purely declarative specification language. The DSL spectrum is very broad, ranging from domain-specific programming to high-level modeling and specification to languages that are not meant to be executable at all.

The WWW is a veritable breeding ground for DSLs. The huge activity in Web software development justifies large investments in new DSLs and their standardization, with the W3C playing a pivotal role. HTML, OWL, XML, WS-BPEL, WS-CDL, WSDL, SMIL, XForms, and UDDI are examples of DSLs directly or indirectly spawned by the WWW. Many of them target aspects of service-oriented computing (SOC), a paradigm that has emerged in the context of the Web.

Using the WWW as an indicator, we expect that DS(M)Ls will play an important role in ULSSIS engineering. This view is shared by the SEI ULS report [3], which explicitly mentions DSLs in Section 6.4.1 (Expressive Representation Languages). We think the potential of DSLs extends to some other sections of the ULS report that do not mention them explicitly. Any ULSSIS will probably give rise to new paradigms, both general-purpose as well as domain-specific ones.

Taking a quick tour through the ULS report, we indicate some (but by no means all) of the applications of DS(M)Ls in ULSSIS engineering, some of the scaling issues involved, and the possibilities for raising expressiveness beyond current levels.

2. THOUSANDS OF PLATFORMS

A platform is the combination of hardware and software that provides a virtual machine that executes software and applications. A ULSSIS will have thousands of platforms [3, Section 1.2]. In principle, platform independence can be achieved by moving to higher abstraction levels (specification, modeling). Platform Independent Models (PIMs) are already current practice in MDE. Given a Platform Definition Model (PDM), a PIM is refined to a Platform Specific Model (PSM). For ULSSIS engineering this approach will have to be be scaled up significantly. We envision that in ULSSISs not only PIM to PSM transformations will be described using DSLs, but also PIM to PIM transformations in support of model evolution.

3. CONTINUOUS EVOLUTION

Continuous software evolution in a ULSSIS [3, Section 2.3] will not only be a problem for code but also for other software artifacts such as specifications, models, and deployment scripts. All these artifacts can be represented using

ULSSIS'08, May 10–11, 2008, Leipzig, Germany.

appropriate DS(M)Ls. Hence, the crucial problem is the evolution of DSLs. What are the current techniques and what can be envisioned for the future? More research is needed towards a more formal incremental approach. For example, in formal verification one has to repeat all verifications whenever new increments are added, not just to verify the increment. How to support evolution at the language level by providing more explicit support for software evolution? Another issue, but completely different, is how DSLs can be easily evolved. Can integration of change be done using the technique known as staging? What about run-time adaptation of a ULSSIS? Moreover, modification of one representation should be reflected in other representations. This problem can be nicely solved with DSLs since all other artifacts (documentation, code) should be automatically derived from the specification expressed in the DSL.

4. EXPRESSIVE REPRESENTATION LANGUAGES

As stated in [3, Section 6.4.1] ULSSIS engineering requires increased language expressivity. The two basic approaches to scaling up expressivity are sacrificing generality and raising the level of abstraction. The former is used by DSLs and the latter by modeling languages, with UML as the prime example. Both mechanisms are used simultaneously by DSMLs, but also by highly declarative DSLs. Like UML, DSMLs often have a graphical syntax, but apart from this difference, there is little to distinguish DSMLs from other highly declarative DSLs, whether graphical or textual.

Both approaches to scaling up expressivity have their limitations. On the one hand, increased specialization leads to an increase in the number of languages needed to cover a given domain, with attendant interoperability problems. On the other hand, the expressivity gained from raising the abstraction level may have to be compensated for by providing important "details" separately. At best, this leads to a beneficial separation of concerns, at worst to collections of software artifacts with hard-to-understand interdependencies. Despite these limitations, we do not think that specialization and abstraction are at the end of their useful life as expressivity scaling mechanisms.

In its present stage, let alone for ULSSIS engineering, the semantics of modeling languages is insufficiently developed [4]. It is usually given by model compilers or generators implemented in general-purpose programming languages. Such a semantics is inadequate from the viewpoint of documentation and verification, and it cannot serve as a basis for automatic generation of language-specific tools such as verifiers, test engines or debuggers. Also, model processing tools may interpret a model differently if DSML semantics is underdefined. In practice, multiple domains may be involved, and the corresponding DSMLs and language-specific tools may have to be composed in some way.

5. SCALED-UP VALIDATION, VERIFICATION, AND CERTIFICATION

DS(M)Ls may help to achieve at least part of the up-scaling of validation, verification, and certification required for ULSSIS engineering [3, Section 6.4.2]. DSLs offer possibilities for analysis, verification, optimization, parallelization, and transformation at a high level in terms of domain-specific constructs that would be much harder or unfeasible if a general-purpose language is used. In the general-purpose case the software artifacts involved would be full of accidental complexities hampering reliable recognition of domain-specific patterns.

6. POLICY-BASED MODIFICATIONS

We see a role for DSLs in formal policy specification, with a view to check policy conformance partly or fully automatically as mentioned in [3, Section 6.5.2].

7. HUMAN INTERACTION

As stated in [3, Section 6.1] research on user centered specifications and on modeling users and user communities is needed. The main issue is how to represent user knowledge and belief systems in such a way that they are comprehensible and analyzable. Again, we see the use of appropriate DSLs as a step toward achieving these goals. Moreover, different GUIs can be generated from user specifications that better suit specific users and their expectations as well as different contexts.

8. ADAPTABLE AND PREDICTABLE SYSTEM QUALITY

The above-mentioned potential of DSLs for high-level specification, analysis, verification, and transformation may also give them a role in specifying, analyzing, and controlling quality attributes of ULSSISs as mentioned in [3, Section 6.6].

9. ORCHESTRATION AND CONTROL

Coordination and control of actions in ULSSISs [3, Section 3.2] can be described using DSLs by specifying general rules of behavior with an underlying adaptable algorithm that govern changing constraints, missions, and functionality of ULSSIS actors. Coordination of artifacts produced by multi-languages or among multi-languages (specification, modeling, programming, representation languages) can be done with DSLs, too. But note that evolution of such multi-language systems is even harder.

10. REFERENCES

[1] J. Gray, J.-P. Tolvanen, S. Kelly, A. Gokhale, S. Neema, and J. Sprinkle. Domain-specific modeling. In P. A. Fishwick, editor, *CRC Handbook of Dynamic System Modeling*, chapter 7. CRC Press, 2007.

[2] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[3] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau. Ultra-large-scale systems: The software challenge of the future. Technical report, Software Engineering Institute, Carnegie Mellon University, 2006. `http://www.sei.cmu.edu/uls/`.

[4] OMG. *Workshop on Precise Behavior Semantics for Domain-Specific Modeling Languages*, Jacksonville, FL, September 25 2007. `http://www.omg.org/cgi-bin/doc?mic/2007-09-01...-09`.