HAL open science

On the Globalization of Domain-Specific Languages

Betty H.C. Cheng, Benoit Combemale, Robert B. France, Jean-Marc Jézéquel, Bernhard Rumpe

To cite this version:

Betty H.C. Cheng, Benoit Combemale, Robert B. France, Jean-Marc Jézéquel, Bernhard Rumpe. On the Globalization of Domain-Specific Languages. Globalizing Domain-Specific Languages, 9400, Springer International Publishing, 2015, LNCS, 10.1007/978-3-319-26172-0_1 . hal-01224105

HAL Id: hal-01224105

https://inria.hal.science/hal-01224105v1

Submitted on 4 Nov 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On the globalization of Domain-Specific Languages

Betty H. C. Cheng[1], Benoit Combemale[2,3], Robert B. France[4], Jean-Marc Jézéquel[2], and Bernhard Rumpe[5]

[1] Michigan State University - East Lansing, US
[2] University of Rennes, FR
[3] Inria, FR
[4] Colorado State University, US
[5] Software Engineering, RWTH Aachen, DE

Abstract. In the software engineering community, research on domain-specific languages (DSLs) is focused on providing technologies for designing languages and tools that enable domain experts to develop system solutions efficiently. Unfortunately, the current lack of support to explicitly relate concepts expressed in different DSLs makes it difficult for software and system engineers to reason about information distributed across models or programs describing different system aspects, at different levels of abstraction. Supporting the coordinated use of DSLs is what we call the *globalization of DSLs*. In this chapter, we introduce a grand challenge of the globalization of DSLs, and we present a few motivating scenarios for such a grand chalenge.

Keywords: Domain-Specific Language, DSL, Globalization of DSLs, Model coordination, Modelling

1 Introduction

The development of current and future complex software-based systems such as avionic, intelligent transportation, smart grid, and smart city and building lifecycle management systems, requires experts from diverse domains to work in a coordinated manner on different aspects of the system. For example, the development of a software system that provides energy-efficient building lifecycle management support for energy-aware development, occupation, maintenance, and demolition of smart buildings, typically requires a system development team that includes experts from a variety of domains, including building architecture, material sciences, environmental sciences, energy management, urban/city/town planning, cybersecurity, software engineering, and sensor networks. Each domain has its own knowledge space that is supported by specialized software languages, techniques, and tools. A major problem facing such development teams is how to bridge the expertise gap between the diverse domains during system development. Communication among the different domain experts is difficult to achieve due to the lack of a common vocabulary and/or mechanisms that effectively relate domain-specific system information expressed in the different models, tools,

techniques, and processes used by the domain experts. Coordination of development activities across the different domains of expertise is particularly necessary when the domains are intertwined, that is, when system decisions made by experts in one domain depends on or influences decisions made by experts in other domains. This type of dependency is common in modern complex systems and can add significant complexity to these systems.

2 Domain-Specific (Modeling) Languages

Model-Driven Engineering (MDE) aims at reducing the accidental complexity associated with developing complex software-intensive systems [8]. A primary source of accidental complexity is the large gap between the high-level concepts used by domain experts to express their problem statements and the low-level abstractions provided by general-purpose programming languages [4]. Manually bridging this gap, particularly in the presence of changing requirements, is costly in terms of both time and effort. MDE approaches address this problem through the use of modeling techniques that support separation of concerns and automated generation of major system artifacts (*e.g.,* test cases, implementations) from models. In MDE, a model describes an aspect of a system and is typically created for specific development purposes. Separation of concerns is supported through the use of different modeling languages, each providing constructs based on abstractions that are specific to an aspect of a system. For example, Generalized Stochastic Petri Nets can be used to create performance models [1], while the notation provided by the Simulink[6] tool is adapted to simulation models. MDE technologies also provide support for manipulating models; for example, there exists tool support for querying, transforming, merging, and analyzing (including executing) models. As such, modeling languages are at the core of MDE.

Incorporating domain-specific concepts and best practices development experience into MDE technologies can significantly improve developer productivity and system quality. This realization has led to work, starting in the late nineties, on MDE-based language workbenches that support the development of domain-specific (modeling) languages (DSLs) and associated tools (e.g., model editors and code generators) [3]. A DSL provides a bridge between the (problem) space in which domain experts work and the implementation (programming) space. Domains in which DSLs have been developed and used include those for automotive, avionics, and cyber-physical systems (CPS). More and more details are also used to describe technical domains, such as configuration of distributed systems and communication networks, deployment structures, mappings of high-level messages to low-level signals, or script languages that guide and control the generation, compilation and deployment processes. It is worthwhile to distinguish technological DSLs and application DSLs, and to recognize that typically several of those DSLs need to be coordinated within a given project.

[6] http://www.mathworks.com/products/simulink

Through an empirical study, Whittle *et al* identified practices and trends that seem to indicate that DSLs can pave the way for wider industrial adoption of MDE [9]. Research on systematic development of DSLs has produced a technology base that is sufficiently robust to support the integration of DSL development processes into large-scale industrial system development environments. Current DSL workbenches support the development of DSLs to create models that play pivotal roles in different development phases.

Workbenches such as Microsoft's DSL tools[7], MetaCase's MetaEdit+[8], JetBrains's MPS[9], Eclipse Modeling Framework (EMF)[10], MontiCore[11] and the Generic Modeling Environment (GME)[12] support the specification of the abstract syntax, concrete syntax and the static and dynamic semantics of a DSL. These workbenches address the needs of DSL developers in a variety of application domains.

3 A Grand Challenge of the Globalization of DSLs: Looking Ahead

The development of modern complex software-intensive systems often involves the use of multiple DSLs that capture different system aspects [2]. In addition, models of the system aspects are seldom manipulated independently of each other. System engineers are thus faced with the difficult task of relating information presented in different models. For example, a system engineer may need to analyze a system property that requires information scattered in models expressed in different DSLs. Current DSL development workbenches provide good support for developing independent DSLs, but provide little or no support for integrated use of multiple DSLs. The lack of support for explicitly relating concepts expressed in different DSLs makes it difficult for developers to reason about information distributed across different models.

Past research on DSLs focused on their use to bridge the wide problem to implementation gap. A new generation of complex software-intensive systems, for example, smart health, smart grid, smart home, intelligent automation, building energy management, and intelligent transportation systems, presents new opportunities for leveraging modeling languages. The development of these systems requires expertise in a variety of domains. Consequently, different types of stakeholders (e.g., scientists, engineers and end-users) must work in a coordinated manner on various aspects of the system across multiple development phases. DSLs can be used to support the work of domain experts who focus on a specific system aspect (e.g., network communication or security), but they

[7] http://www.microsoft.com/en-us/download/details.aspx?id=2379
[8] http://www.metacase.com/fr/mwb/
[9] https://www.jetbrains.com/mps
[10] http://www.eclipse.org/modeling/emf
[11] http://www.monticore.de
[12] http://www.isis.vanderbilt.edu/projects/gme/

can also provide the means for coordinating work across teams specializing in different aspects and across development phases.

Supporting coordinated use of DSLs leads to what we call the *globalization of DSLs*, that is, the use of multiple DSLs to support coordinated development of diverse aspects of a system. We can make an analogy with globalization in the real world, in which relationships are established between sovereign countries to regulate interactions (e.g., travel and commerce related interactions), while preserving each country's independent existence. The term "DSL globalization" is used to highlight the overarching objective that DSLs developed in an independent manner to meet the specific needs of domain experts should also have an associated framework that regulates interactions needed to support collaboration and work coordination across different system domains.

Globalized DSLs are intended to support the following critical aspects of developing complex systems: communication across teams working on different aspects, coordination of work across the teams, and well-defined management of the teams to ensure product quality. In the vision for globalized DSLs, integrated DSLs support teams working on systems that span many domains and concerns to determine how their work on a particular aspect influences work on other concerns. The objective is to offer support for communicating relevant information, and for coordinating development activities and associated technologies within and across teams. In addition, globalized DSLs should provide support for imposing control over development artifacts produced by multiple teams.

Coordination and related separation of concerns issues have been the focus of software engineering since early work on modularizing software [7]. For example, Parnas' use of the term "work product" to denote a module that can be the source of independent development is also a focus of team demarcation across design and implementation tasks. Modularity in modern software-intensive systems development leads to well-known coordination problems, such as problems associated with coordinating work over temporal, geographic or socio-cultural distances [6]. This line of work has also led to the recognition of socio-technical coordination, including coordination of the stakeholders and the technologies they use to perform their development work, as a major system development challenge [5].

In this context, DSLs can be used to support socio-technical coordination by providing the means for stakeholders to bridge the gap between how they perceive a problem and its solution, and the programming technologies used to implement a solution. DSLs also support coordination of work across multiple teams when they are supported by mechanisms for specifying and managing their interactions. In particular, proper support for coordinated use of DSLs leads to language-based support for social translucence, where the relationships between DSLs are used to extract the information needed to make teams working on different aspects of the system aware of the the project activities from other teams. Such awareness is needed to minimize the counter-productive form of social isolation that can occur when work is distributed across different teams.

4 Motivating Scenarios for the Globalization of DSLs

We now discuss several motivating scenarios for the globalization of DSLs. For each, we describe the typical scenarios encountered by engineers that lead to the need for globalization and show the impact on the overall globalized ecosystem.

Global System Checking : The need for the globalization of DSLs first arises when a system engineer wants to assess a system property that requires crosscutting information scattered in various models. In such a case, system engineers face the difficult task to either build a global structural or behavioral specification of the system from the various models to be able to check the global property or to enhance coordination techniques at hand that enable coordinated models to be checked for global properties.

Model Consistency Checking : In complex software intensive systems where different intertwined DSLs are used to describe the models of the various aspects of the same system, evolving a DSL or a model may have important consequences on the system design as a whole. Since the models of the different system aspects are seldom manipulated in isolation, the development of a model expressed in one DSL can directly influence the form of models created using other DSLs. Similarly, if the different DSLs used for different aspects of a system are tightly coupled, then it is likely that evolving one of them will impact the others. In both cases, syntactic and semantic consistency relationships defined across the DSLs can be used to ensure that the different models and DSLs are consistent with one another.

Traceability for Impact Analysis : As a particular case of consistency checking, one may analyze the impact of a change in one model with respect to other models. For instance, when a requirement changes, one may evaluate the impact on the entire system design. In such cases, traceability links between the various models built all along the development process are required.

Language Evolution : By definition, DSLs evolve as the concepts in a domain and the expert understanding of the domain evolves. As such, it is essential to address consistency between models and DSLs when the DSL specifications change. As a DSL evolves, the conforming models need to evolve accordingly in order to remain consistent with new constructs, new constraints, or changes in the semantics. These consistency demands might lead to a snowball effect, where all the tools, transformations, or workbenches defined around a language need to be updated. In typical large projects, neither all languages nor all models of these languages are evolved in parallel. Therefore, it is necessary to coordinate the parallel use of models in different variants of the same language as well.

Model Composition : Separation of concerns is achieved in MDE by defining as many models as concerns of the system. Eventually, all the different models must be composed in order to support, for example, the generation of the entire system implementation. When different DSLs are used to define the various models, composition rules must be defined between the DSLs.

Simulation : Unfortunately, a simulation of a substantial part of the real world needs to describe different parts and aspects of the world typically using several languages. To run simulations, we need a stable coordination of languages and their respective models for execution. This coordination enables us to understand, for example, whether the models fit together and whether they correctly describe the real world and system to be designed. Examples for coordinated model simulation can be found in various domains, including climate that models whether flow of water, cultivation of areas, run in parallel, and etc. Other simulations are used to understand how control devices in a car cooperate or how the multitude of existing devices in an airplane can be managed by pilots for example.

References

1. Gianfranco Balbo. Introduction to generalized stochastic petri nets. In Marco Bernardo and Jane Hillston, editors, *Formal Methods for Performance Evaluation*, volume 4486 of *Lecture Notes in Computer Science*, pages 83–131. Springer Berlin Heidelberg, 2007.
2. Benoit Combemale, Julien Deantoni, Benoit Baudry, Robert France, Jean-Marc Jézéquel, and Jeff Gray. Globalizing Modeling Languages. *Computer*, pages 68–71, June 2014.
3. Sebastian Erdweg, Tijs van der Storm, Markus Völter, Meinte Boersma, Remi Bosman, William R. Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, Gabriël D. P. Konat, Pedro J. Molina, Martin Palatnik, Risto Pohjonen, Eugen Schindler, Klemens Schindler, Riccardo Solmi, Vlad A. Vergu, Eelco Visser, Kevin van der Vlist, Guido Wachsmuth, and Jimi van der Woning. The state of the art in language workbenches - conclusions from the language workbench challenge. In *Software Language Engineering - 6th International Conference, SLE 2013, Indianapolis, IN, USA, October 26-28, 2013. Proceedings*, volume 8225 of *Lecture Notes in Computer Science*, pages 197–217. Springer, 2013.
4. Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In Lionel C. Briand and Alexander L. Wolf, editors, *Proceedings of the Future of Software Engineering Symposium (FOSE '07)*, pages 37–54. IEEE, July 2007.
5. James D. Herbsleb. Global Software Engineering: The Future of Socio-technical Coordination. In Lionel C. Briand and Alexander L. Wolf, editors, *Proceedings of the Future of Software Engineering Symposium (FOSE '07)*, pages 188–198. IEEE, July 2007.
6. J.D. Herbsleb and R.E. Grinter. Architectures, Coordination, and Distance: Conway's Law and Beyond. *IEEE Software*, pages 63–70, 1999.
7. D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, December 1972.
8. D.C. Schmidt. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31, 2006.
9. Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *Software, IEEE*, 31(3):79–85, 2014.