Evaluating Domain-Specific Modelling Solutions

Parastoo Mohagheghi, Øystein Haugen

SINTEF, Forskningsveien 1, Oslo, Norway
{parastoo.mohagheghi, oystein.haugen}@sintef.no

Abstract. This paper presents criteria and evaluation methods for evaluating domain-specific modelling (DSM) solutions based on analysing state of the art and experiences of developing and evaluating DSM solutions in research projects. The state-of-the-art analysis returned several requirements regarding the quality of domain-specific modelling languages and tools developed based on them that are classified based on the identified stakeholders. The stakeholders are those who develop and those who use a DSM solution, the intended domain and purposes with developing a DSM solution as defined by domain experts, software engineering concerns, integration with other languages or tools, and the quality of artefacts to be modelled or generated. Both quantitative and qualitative approaches may be applied for evaluating DSM solutions based on the development stage and requirements. There is a clear need for a process that supports evaluating the quality of DSM solutions and this research contributes to the definition of such process.

Keywords: domain-specific language, modelling, assessment, quality, case study

1 Introduction

General-purpose modelling languages like UML are already widely used in industry, but the experience of many cases shows that learning and adopting them to specific contexts is difficult, such that they are not always the best fit for solving special problems. This is the reason why domain-specific modelling is receiving attention by industry, also because the domain-specific modelling environments are getting more powerful and mature.

A Domain-Specific Language (DSL) is typically a small, highly focused language used to model and solve some clearly identifiable problems in a domain; in contrast to a General-Purpose Language (GPL) that is supposed to be useful for multiple domains. DSLs may operate stand alone, be called at run-time from other programs or be embedded into other applications to do specific tasks. DSLs may be designed from scratch or by extending a base language (e.g., defining profiles in UML). Mernik et al. discuss different approaches to the development of DSLs and their advantages and disadvantages and also write that DSL development is hard, requiring both domain knowledge and language development expertise [9]. Besides, it is often far from evident that a DSL might be useful or that developing one might be worthwhile.

Several domain-specific modelling languages (DSML) and editors and transformations for modelling, generation and other purposes such as simulation (generally referred as DSM solutions) have been developed in the context of four industrial partners involved in the European IST project MODELPLEX[1]. MODELPLEX aimed at applying Model-Driven Engineering (MDE) techniques on scenarios of complex software systems. The industrial domains here were enterprise business applications, telecommunication, aerospace crisis management systems and data intensive geological systems. Examples of DSM solutions developed in MODELPLEX are a network modelling tool and DSMLs for security and performance engineering. Also a DSM solution for specifying signalling at railway stations and generating source code has been developed in the ITEA-MoSiS[2] project. All of these DSM solutions are meant to be used by domain experts and thus should be understandable by these experts. Some questions that arose regarding the quality of these DSM solutions were:

- Is the DSM solution easily usable by the intended domain experts?
- Does the DSM solution provide appropriate built-in abstractions and notations for building applications in the specific domain?
- Does the DSM solution serve the purpose of the development such as generating relevant artefacts?
- Is the DSM solution maintainable and evolvable when the domain evolves?
- Is the DSML small enough, leaving out language features that do not contribute to the purpose of the language?

In order to apply a systematic approach for evaluating DSM solutions, we performed a state-of-the-art analysis on evaluating languages used in software development in general and DSLs in particular. The analysis identified several characteristics of DSLs and DSMLs that showed to be relevant for our work. We also detected a few examples of evaluation. This paper summarizes the results of this analysis, discusses experiences of evaluating DSM solutions in the research projects MODELPLEX and MoSiS, and proposes directions for future work.

The remainder of this paper is organized as follows. Section 2 presents the identified evaluation criteria and a classification of them while Section 3 focuses on evaluation methods. Section 4 is the discussion of two case studies. Finally, the paper is concluded in Section 5 and future work is discussed.

2 Criteria for Evaluating Domain-Specific Modelling Languages

Related work can be discussed in several dimensions: evaluating languages in general, evaluating modelling languages, and evaluating domain-specific languages. We focus on the last two while some general characteristics of languages relevant for our discussion are also included.

[1] MODELPLEX- MODelling solutions for comPLEX software systems (2006-2010); http://www.modelplex.org/

[2] MoSiS- Model-driven development of highly configurable embedded Software intensive Systems (2007-2010); http://itea-mosis.org/modules/wikimod/index.php?page=WikiHome

Howatt proposes four classes of criteria for evaluating languages [4]:

- *Language Design and Implementation Criteria*: Is the language formally defined? Can a fast, compact compiler be written to generate efficient, compact code?
- *Human Factors Criteria*: These criteria are used to assess the human interface or the user-friendliness of a language.
- *Software Engineering Criteria*: These assess those aspects of a language that enhance the engineering of good software; for example supporting portability, reliability and maintainability of the software.
- *Application Domain Criteria*: These criteria assess how well a language supports programming for specific applications.

Kennedy et al. add two other criteria to this list [7]: The *time and effort* required to write, debug, and tune the code, and the *performance* of the code that results.

Lindland et al. describe their framework for evaluating conceptual models in [8]. Conceptual models are models developed in early phases of development. The framework defines three quality goals for models:

- *Syntactic quality* is how well the model corresponds to the language,
- *Semantic quality* is how well the model corresponds to the domain,
- *Pragmatic quality* is how well the model corresponds to its audience interpretation.

The Lindland et al.'s framework distinguishes between quality goals and means to achieve the goals. For example, having a formal syntax helps to achieve syntactic quality.

Grossman et al. use the following criteria and those identified in [1] for evaluating UML in [2]. The criteria are however mostly relevant for DSLs as well:

- *Having right data* which are necessary constructs and their semantics. *Completeness* is added in [15], which is capturing all concepts.
- *Accuracy of concepts* to present the developed system and helping in designing it.
- *Flexibility* to model different systems and ease of change.
- *Understandability* in the ease of read and conveying the meaning of the underlying system.
- *Level of detail and needed training*.

Paige et al. have also identified some principles in the design of modelling languages that may be used as evaluation criteria for evaluation DSMLs [11]. Examples are:

- *Simplicity*: no unnecessary complexity, including being small and memorable.
- *Uniqueness or orthogonality*: no redundant or overlapping features.
- *Consistency*: language features cooperate to meet language design goals.
- *Seamlessness*: mapping concepts in the problem space to implementations in the solution space, and the same abstractions can be used throughout development.
- *Space economy*: concise models are produced.

An analysis of the identified characteristics shows that these are defined from multiple viewpoints by different stakeholders. Based on the covered literature, we have identified the stakeholders interested in a DSM solution and classified the identified criteria according to their interests as depicted in Fig.1.

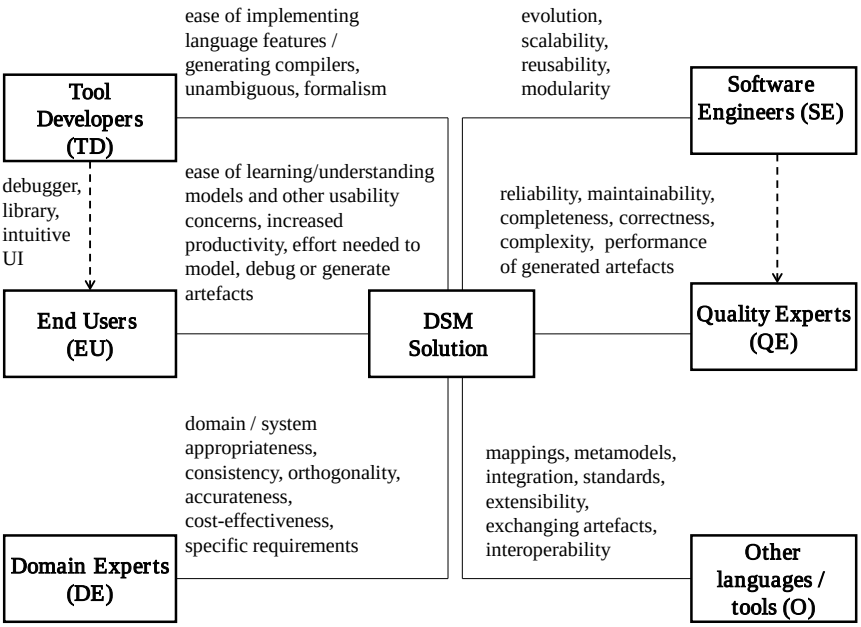

Fig. 1. Evaluating a DSM solution by different stakeholders

The stakeholders are defined below and examples of criteria of interest for them are discussed:

- *Tool Developers (TD)* are those developing the DSML and related tools. Examples of relevant criteria for them are those identified by Howatt as language design and implementation criteria and application domain criteria [4].
- *End-Users (EU)* are those using the DSM solution for modelling or generating artefacts. Usability and ease of learning are examples of criteria relevant for them. The link between *TD* and *EU* suggests that providing some support by tool developers such as including a useful library, debugger and an intuitive User Interface (UI) helps improving end-users' experience with the DSM solution.
- *Domain Experts (DE)* represent the domain of interest and the purpose of a DSM solution. In general, a DSML should include appropriate domain concepts and abstractions [9] and be complete and accurate. A DSM solution may be developed for multiple purposes such as programming directly in the terms used by domain experts and thus reducing the gap between domain experts and software developers, automating software development or improved quality of the code. The evaluation should therefore focus on the purpose of a DSM solution.
- *Software Engineers (SE)* are interested in the characteristics of the DSM solution that lead to developing good software. Examples of their concerns are reuse of models and evolvability of the DSM solution. Applying some software engineering practices also improve the quality of models and generated artefacts.

- *Quality experts (QE)* are interested in the quality of models or artefacts generated from models. These may have requirements regarding completeness and performance of the generated code, its completeness and even understandability of models and generated artefacts for maintenance.
- *Other languages / tools (O)* cover requirements for interoperability with other tools or languages, mappings between languages or tools, building extensions, and compliance to standards if required. There are several approaches for developing a DSML (such as developing from scratch or extending an existing language) and the *O*-characteristics should be considered when selecting the approach.

The model depicted in Fig.1 allows classifying identified criteria in a meaningful way and is applied when selecting evaluation criteria in the case studies discussed in Section 4. The identified evaluation methods are discussed in the next section.

3 Evaluation Methods

To perform the evaluation of a DSM solution, one may take advantage of quantitative or qualitative approaches. For *quantitative evaluation*, some identified metrics are:

- *Time and effort required to model, debug, and generate artefacts* (from [7]). We may also add *time and effort to understand models*. One may compare time and effort when using a DSM solution with time and effort without using a DSM solution in a controlled experiment as done in [5].
- *Performance* of *the code that results from models* (from [7]).
- *Collecting metrics from models* such as the number of model elements. Model metrics are discussed in [10]. A large amount of metrics can be defined on models while identifying useful model metrics is a challenge.
- *Usability metrics* are discussed in [14]. Seffah et al define usability as "whether a software product enables a particular set of users to achieve specific goals in a specific context of use" and covers efficiency, productivity, satisfaction, learnability, safety and usefulness for solving problems. Some proposed metrics are time to learn or perform tasks, user steps to perform a task and layout appropriateness.
- *Number of concepts and the relations between these concepts in the DSML* [13]. This metric is on the metamodel of languages and assumes that languages with more concepts and relations are more complex, such as UML. Since DSMLs are usually small languages, this count will probably not return interesting information.
- *Evaluating metamodel's understandability by performing controlled experiments* as discussed in [12]. Both syntactic understanding which refers to the constructs of the metamodels and relationships (for example, how many attributes describe an employee) and semantic understanding that assess the understanding of contents (for example whether every employee has a unique employee number) are of interest to assess.
- *Performing a survey among users* can generate quantitative data.

Qualitative approaches cover case studies (including comparative ones that compare using a DSM solution with other approaches), analysis of a language and the DSM solution by experts for various characteristics, and monitoring or interviewing users.

A DSM solution may be evaluated both quantitatively and qualitatively. The important issue is to decide which approach is best in which phase of the development lifecycle. The ISO 9126 standard divides metrics in *internal* (design time), *external* and *quality in use* metrics which indicates that properties should be measured in different stages and some design-time measures can be used as prediction of run-time characteristics. Seffah et al. discuss *predictive* and *testing* metrics where predictive metrics may provide an estimate of system usability [14]. Testing metrics are collected when a software product is in use. Kelly and Tolvanen recommend an incremental and test-driven approach for developing DSLs [6]. For a DSM solution, there is often a prototyping phase and a usage phase. In the prototyping phase, evaluation is often done by language experts and pilot users who try the language on small cases. We developed a set of questions for this phase based on the requirements of case studies that is presented in the next section. The evaluation in the prototyping stage is often qualitative. In the usage stage, more users are involved which allows running experiments or collecting opinion of users in a survey.

4 Case Studies

4.1 Evaluating the Network Modelling Tool

The first case discussed here covers developing a network modelling tool in Telefónica using Eclipse GMF. The experiences are discussed in detail in [3]. The key driver for this DSM solution is the recognition that it is becoming increasingly difficult to manage the complexity and size of modern telecom networks. By Telefónica's requirement, the Network DSML had to include specific elements required for modelling and also allow modelling at different levels of abstraction, at least showing the internal of devices, how devices connect to each other and higher-level interactions and roles of whole sub-networks in the deployment of a service. From these models, a wide range of artefacts could be generated such as device configuration specifications.

Rather than developing a metamodel from scratch, a metamodel based on Common Information Model (CIM)[3] was used in this development. CIM was relevant as it is the underlying model in many products dealing with management and instrumentation of network equipment. Finally, there were a number of generic features which were required in order to meet the needs of end-users of the tool. These included: a) a visual, user-friendly interface; b) scalability – enabling thousands of model elements to be managed; c) interoperability with other tools and standards; d) flexibility – enabling the rapid adaptation of the tool to support new abstractions (preferably done by the engineers themselves); and e) support for model validation and checking.

[3] Common Information Model Website, http://www.dmtf.org/standards/cim

The evaluation of the DSM solution was performed by answering a set of questions defined by a team of researchers and domain experts based on the requirements. The feedback by a team of pilot users is based on using the tool for modelling and the generator to produce the required artefacts in some example scenarios. The set of questions from various viewpoints and the results of evaluation are summarized in Table 1.

Table 1. Evaluating the network modelling solution

Stake holder	Question	Results
EU	Is the DSML tool easy to use? Is the UI acceptable?	Not enough, largely due to the sheer size of the metamodel which resulted in having to add a large number of connection and node tools.
EU	Do you intend to use the DSM solution in future projects and invest on making a more usable version?	We would like to use but there are several barriers: The DSML should be smaller and more focused, other tools than GMF for developing it should be evaluated and the DSM solution should be used in a series of projects to investigate Return-On-Investment.
EU	Does the DSM solution affect the performance of users?	Yes, the DSM solution has the potential to improve productivity and quality but additional work and training are needed to achieve those objectives.
EU	Do we think that using the DSM solution improves our reputation and image as innovative?	Yes, the image and reputation of innovation can be greatly improved by the use of tools and approaches such as the one presented herein.
SE	Is the DSM solution scalable?	GMF does not scale well because of some shortcomings in the implementation.
SE	Is the DSM solution flexible?	The same applies to flexibility. A more dynamic, metamodel-driven tool generation approach is needed.
SE	Does the DSM solution provide reuse possibilities?	Modelling at different abstraction levels is applied to increase reusability of elements.
DE	Is the CIM metamodel suitable for modelling network management in Telefónica?	Yes, they are suitable for this purpose but need constant revision and extension to keep up with the evolution of the domain and the standard of reference (CIM).
O	Is the DSML compatible with the standards?	Yes, using CIM provides such compatibility but brings problems due to its size.
O	Is the DSML compatible with other tools?	Many tools used in the network management domain are based on CIM, but as the DSML transforms the CIM metamodel into EMF, this leads to compatibility issues with CIM-based off-the-shelf products that need to be resolved.

One of the most challenging aspects of this DSM solution was the large number of modelling abstractions and relationships in the CIM model. Another challenge was that of making the tool as usable as possible, which involved changing the tooling definition. We experienced that developing a DSML in an environment such as Eclipse required high language and tool expertise, which make developing DSM

solutions out of reach of domain experts with some IT expertise, and the resulting DSM solution is not changeable or flexible enough. Changes to the metamodel which happen frequently in the domain required considerable effort in updating the tool and the developed models became corrupted due to changes.

4.2 Evaluating the Train Control Language

The Train Control Language (TCL) is a DSML for specifying the signalling at railway stations and generating interlocking source code that is used in allocating routes to trains. Using TCL has several benefits compared with the current development process. In the current workflow, errors in the various steps are possible due to the manual procedure. Thus validation of each step is required to ensure the safety of the system. Using TCL most of these steps are automated. By assuring that TCL and the generators are correctly implemented, consistency between the representations can be guaranteed. Therefore some of the validation steps can be eliminated. Several constraints are defined to assure that stations are correctly created, and the editor makes sure that every necessary condition is taken into consideration. If the constraints are properly defined, the TCL tool may guarantee completeness by requiring all necessary elements. Other benefits are implementing a target environment that includes generators such as code generators and analysis tool that prevent or detect inconsistencies or errors in models. Together these benefits lead to significant productivity improvements.

The first step in evaluating the TCL has been identifying stakeholders and their reasons for developing a DSM solution. We identified the stakeholders to be: a) *tool developers* who have developed the metamodel and supporting tools; b) *signalling engineers* who are the end-users that will model the stations; c) *station deployers* that will generate required source code; d) *testers* who will generate test cases from the models; and e) *railway authorities* who are the standardization organs and national authorities that define safety requirements. The second step in evaluation has been identifying quality requirements of these stakeholders. Finally, we have also identified evaluation method for each requirement, and how a requirement can be achieved by "means" that should be applied. Examples of quality requirements, means and evaluation methods are depicted in Table 2.

Table 2. Examples of requirements for evaluating TCL, means and evaluation methods

Stake holder	Requirement	Means	Evaluation Method
EU	TCL models should be similar to existing diagrams.	Walking through existing examples together with Station deployers.	Performing visual comparison of models developed with the first version of TCL with existing diagrams.
EU	Small stations should be covered completely.	Small stations are identified and their models are reviewed.	Models developed with the first version of TCL are compared with existing diagrams.

Stake holder	Requirement	Means	Evaluation Method
EU	TCL and tools should prevent specifying unsafe models.	Adding well-formedness rules to the language. Also adding constraints to the TCL specifications.	Validate the constraints by inspections and running test cases.
O	Models are compliant with safety standards.	Add constraints to the TCL specifications. Also, integrate the safety standards in the necessary steps in the development process.	Validate the constraints and inspect the development process.

The actual evaluation of the language as identified by evaluation methods remains to be performed. At this stage, the evaluation work has helped the involved stakeholders to clarify and communicate their intentions with the DSM solution, the implemented features of the solution (defined as means) and relating requirements to features.

5 Conclusions and Future Work

The quality of domain-specific languages (DSLs) and modelling solutions has been subject of some research by now. Based on a state–of-the-art analysis and experiences with developing domain-specific modelling (DSM) solutions in research projects, we have identified several evaluation criteria. These are currently classified according to the stakeholders interested in them. We have also identified evaluation methods and examples of evaluation. All these are included in a framework for evaluating DSM solutions which is under development and should include examples of best practices or means as well.

Based on the experiences so far, we can summarize that some characteristics are especially important for DSLs. An important criterion is domain-appropriateness. A DSL must be powerful enough to capture the major domain concepts and should match the mental representation of the domain. DSM solutions are typically used for prediction or simulation, as well as code generation, test generation and execution. Thus the language should be formal and accurate. Any DSL with a diagrammatical syntax should have proper layout, and the there is often a need for integrating DSLs with other ones. Performing a systematic review of published literature for identifying all related research will contribute to this work.

We have also performed several case studies on evaluating DSM solutions in the early phase of development using a questionnaire. We presented two cases of evaluation in this paper. Relating evaluation criteria to evaluation methods is also subject for future work.

When discussing DSM solutions, it is of key importance to focus on the needs of an often narrow application domain and the actual purposes of the DSM solution. The development of a DSM solution is iterative and so is the assessment. Having the

requirements in mind, there is a clear need for a process that supports defining and evaluating the quality of domain-specific solutions. We have identified some steps of this process as identifying stakeholders and requirements of the DSM solutions, identifying means to achieve the requirements, and identifying evaluation methods. We will continue work on this process in future work.

Acknowledgments. This work has been supported by the MODELPLEX project (IST-FP6-2006 Contract No. 34081) and the MoSiS project ITEA 2 – ip06035.

References

1. Goodhue, D.L.: Development and Measurement Validity of a Task Technology Fit Instrument for User Evaluations of Information Systems. Decision Sciences 29 (1), pp.105—138 (1998)
2. Grossman, M., Aronson, J.E., McCarthy, R.V.: Does UML Make the Grade? Insights from the Software Development Community. Information and Software Technology 47, pp. 383—397 (2005)
3. Evans, A., Fernández, M.A., Mohagheghi, P.: Experiences of Developing a Network Modelling Tool Using the Eclipse Environment. In ECMDA-FA 2009, LNCS, vol. 5562, pp. 301–312, Springer (2009)
4. Howatt, J.: A Project-Based Approach to Programming Language Evolution. URL http://academic.luther.edu/~howaja01/v/lang.pdf, visited in August 2007 (2001)
5. Kärnä, J., Tolvanen, J.P., Kelly, S.: Evaluating the Use of Domain-Specific Modeling in Practice. In 9th OOPSLA Workshop on Domain-Specific Modeling (2009)
6. Kelly, S., Tolvanen, J-P: Domain-Specific Modeling- Enabling Full Code Generation. IEEE Computer Society Publications (2008)
7. Kennedy, K., Koelbel, C., Schreiber, R.: Defining and Measuring the Productivity of Programming Languages. In: International Journal of High Performance Computing Applications, Volume 18, Issue 4, pp. 441—448 (2004)
8. Lindland, O.I., Sindre, G., Sølvberg, A.: Understanding Quality in Conceptual Modelling. IEEE Software 11 (2), pp. 42—49 (1994)
9. Mernik, M., Heering, J., Sloane, A.M.: When and How to Develop Domain-Specific Languages. ACM Computing Surveys 37(4), pp. 316—344 (2005)
10. Mohagheghi, P., Dehlen, V.: Existing Model Metrics and Relations to Model Quality. In 2009 ICSE Workshop on Software Quality (WoSQ'09), IEEE CS, pp. 39-45 (2009)
11. Paige, R.F., Ostroff, J.S., Brooke, P.J.: Principles for Modeling Language Design. Information and Software Technology 42, pp. 665--675 (2000)
12. Patig, S.: Preparing Meta-Analysis of Metamodel Understandability. In Workshop on Empirical Studies of Model-Driven Engineering (ESMDE'08), pp. 11--20 (2008)
13. Rossi, M., Brinkkemper, S.: Complexity Metrics for System Development Methods and Techniques. Information Systems 21(2), pp.209—227 (1996)
14. Seffah, A., Donyaee, M., Kline, R.B., Padda, H.K.: Usability Measurement and Metrics: a Consolidated Model. Software Quality Journal (14), pp. 159--178 (2006)
15. Teeuw, W.B., van den Berg, H.: On the Quality of Conceptual Models. URL http://osm7.cs.byu.edu/ER97/workshop4/tvdb.html (1997)