# Automata Based Method for Domain-specific Languages Definition

Ulyana Tikhonova,

Supervisor: Fedor Novikov
St. Petersburg State Polytechnical University, Applied Mathematics Dept.,
Politekhnicheskaya 29., 195251 St. Petersburg, Russia
ulyana.tihonova@gmail.com, fedornovikov@rambler.ru

**Abstract.** We outline a research proposal which goal is to contribute to methods of new Domain-Specific Languages (DSLs) definition and implementation. We propose the automata based method for DSLs definition that allows specifying new languages with various notations in such a way that the language definition can be treated as a ready-to-use language implementation already.
The automata based method allows defining language by three components: language metamodel (which includes an abstract syntax), concrete syntax and operational semantics. We use Unified Modeling Language (UML) as description formalism for all three components. Namely, language metamodel is defined using class diagrams. Concrete syntax is defined as parser using state machine diagrams. Semantics is defined as metamodel interpreter using state machine diagrams as well.

**Keywords:** DSL, metamodel, concrete syntax, operational semantics, UML.

## 1 Motivation

Domain-specific Languages (DSLs) are considered to be very effective in software engineering. They raise the level of abstraction, provide common domain notation, and improve development process therefore. Per se, DSLs allow describing a problem solution in terms of the field of the problem, rather than in terms of computer. The most critical part of the whole software development process in the context of language oriented programming paradigm is definition and implementation of a new DSL [14]. All approaches to solve this problem could be divided into those, which use traditional grammars, and those, which are based on modeling in context of Model Driven Engineering (MDE) [4]. The former approach allows defining programming language as combination of its structure and textual syntax. The latter implies definition of language metamodel mostly in terms of MOF and subsequent usage of the model for code generation, model transformation, etc. In this case, concrete syntax is usually graphical or even undefined.

However, these two styles of language definition do not differ in essence. As was noticed in [5] and [3] the program in any DSL is an abstract structure, and for its editing, storage and execution various representations might be used: text, diagrams, tables, formulas, sounds, etc. Therefore, a single method for definition of different notations is desirable.

Another issue is specification of language semantics. The brief overview of methods of semantics definition is given in [2]. They vary from complicated formulas of axiomatic semantics to rewriting rules of translation semantics. We consider that usage of the same formalism for concrete syntax definition and for semantics definition would simplify the process of new DSL creation, which is rather complicated now.

At last, specification of new DSL should be sufficient for receiving its implementation automatically.

## 2 Brief Overview of Related Work

A number of language workbenches, which support development of DSLs with not only textual notation, were worked out recently. One of them is MetaEdit+ tool [8] that allows creation of graphical DSLs with facility to specify generation of various sorts of target data from DSL diagrams. MetaEdit+ uses its own model of DSL abstract syntax – the metamodeling language GOPPRR (Graph-Object-Property-Port-Role-Relationship).

Another one, AMMA [1], is a model based framework that supports DSL development with metamodel definition language KM3 [11], language for specifying textual concrete syntaxes TCS and model transformation language ATL. This project is based on MOF formalism and supports graphical syntax through class diagrams of models. The common approach is implemented in MOFLON [10] project. In addition the latter uses Story Driven Modeling (SDM) paradigm for definition of the dynamic semantics of a DSL.

We appeal for possibility to define various notations using single technique and for possibility to define both concrete syntax and semantics using the same specification (meta)language.

## 3 Proposed Solution

In this work, we propose just another DSL definition method based on model driven architecture (MDA) and executable UML approach [6]. Definition of DSL using MDA approach instead of traditional formal grammars advances software engineering unification. This approach requires separation of language definition levels, which are abstract syntax, concrete syntax and semantics. Therefore, the proposed method consists in correlated definitions of DSL metamodel (which includes abstract syntax), concrete syntax and operational semantics (Fig. 1). We use UML [12] as description formalism and investigate definition methods that differ from those listed in part 2 for all three steps of DSL specification process.

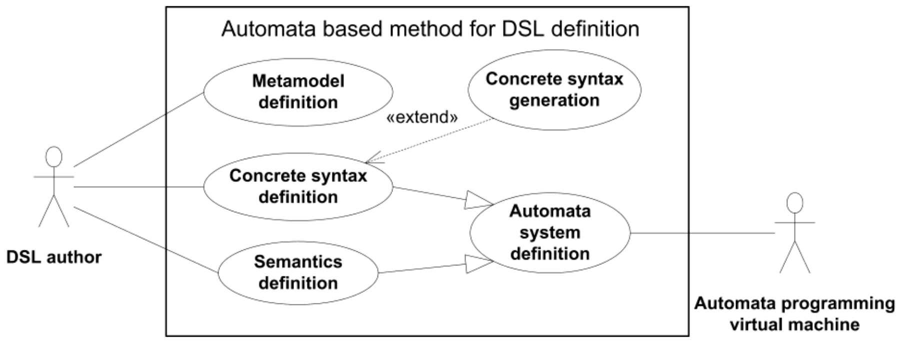
Automata based method for DSL definition
Metamodel definition
Concrete syntax generation
«extend»
Concrete syntax definition
Automata system definition
Semantics definition
DSL author
Automata programming virtual machine

**Fig. 1.** Use case diagram of automata based method for DSL definition

We extend formalism used for language abstract syntax definition to UML class diagram in comparison with widely spread MOF, as UML class diagram can express more copious structures. We propose description both of concrete syntax and of operational semantics as algorithms through UML state machine diagrams. Namely, concrete syntax is defined as a parsing algorithm, which analyzes source program representation and constructs abstract program. Operational semantics is defined as an algorithm of abstract program interpretation.

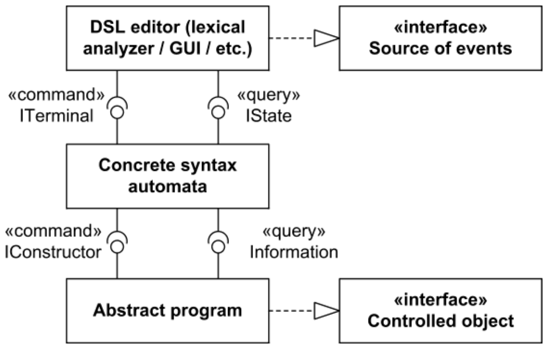

**Fig. 2.** Component diagram of concrete syntax automata, abstract program and DSL editor

One of the key features of the proposed method is unified view on different language notations. Any abstract program representation is considered as chain of events, which are processed by automata system defined in concrete syntax specification. Each terminal representation is considered as an event sent by some source of events. For example, events might be typographical characters in text, geometrical figures in diagram, controls in dialogue window, cells in spreadsheet, or sounds of spoken commands. Accordingly, any source of events is acceptable: a lexical analyzer of text, a graphical editor of diagrams, an editor of formulas or a dialogue window (Fig. 2).

To achieve automatic receiving of DSL's implementation we use automata based programming paradigm [7, 13]. According to this paradigm, every algorithm described through UML state machine diagrams can be executed by an automata programming virtual machine (Fig. 1).

## 4 Research Method

We have developed an initial candidate DSL meta-metamodel – the abstract syntax of the proposed method (Fig. 3). This meta-metamodel accumulates expressiveness both of grammar formalism and of UML class diagram. We are going to investigate it and compare it with formal grammars to find out the kind of languages that could be defined as instances of this meta-metamodel. Moreover, mapping between DSL meta-metamodel and formal grammars could be useful for the reuse of already defined languages. This mapping could be also useful for development of the algorithm of generation of concrete syntax automata system from DSL metamodel.

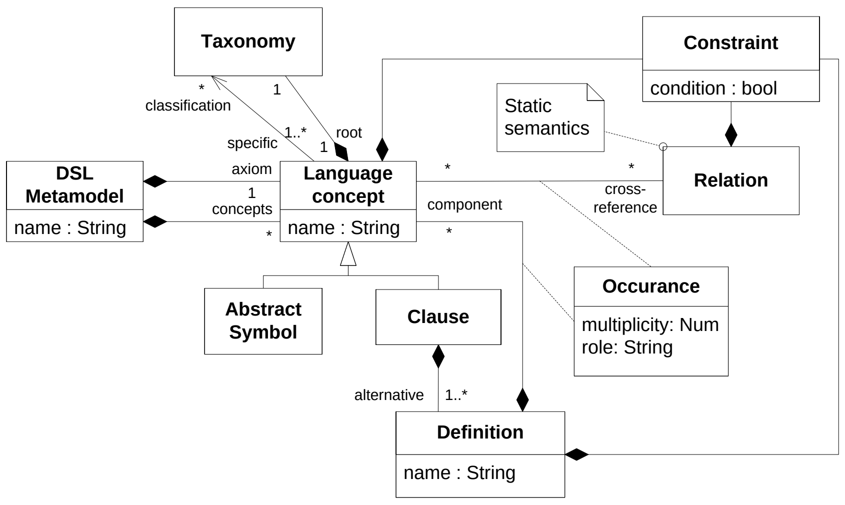

**Fig. 3.** Class diagram of DSL meta-metamodel

We have devised the automata model being used as formalism for definition of both concrete syntax and semantics. Two little DSLs have been specified using this model: nonlinear and graphical language of the chess position and the mini language for manipulations with sets. Further, the automata programming virtual machine should be developed to execute these specifications of DSLs. Definition of the automata programming virtual machine with the means of the proposed method would be the best use case. In other words, we are going to apply the idea of bootstrapping.

The proposed method for DSL definition allows defining different languages with various concrete representations. DSL definition can be used as its software implementation in terms of the automata based virtual machine.

## References

1. Bézivin, J., Jouault, F., Kurtev, I., Valduriez, P.: Model-Based DSL Frameworks. In: 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pp. 602-616. ACM, New York (2006)
2. Combemale, B., Crégut, X., Garoche, P.-L., Thirioux, X.: Essay on Semantics Definition in MDE - An Instrumented Approach for Model. In: Journal of Software, Vol 4, No 9, pp. 943-958 (2009)
3. Dmitriev, S.: Language Oriented Programming: The Next Programming Paradigm, http://www.onboard.jetbrains.com/is1/articles/04/10/lop/index.html (2005)
4. Estublier, J., Vega, G., Ionita, A.D.: Composing Domain-Specific Languages for Wide-Scope Software Engineering Applications. In: Briand, L., Williams, C. (Eds.) International Conference on Model Driven Engineering Languages and Systems (MoDELS). LNCS vol. 3713, pp. 69-83. Springer-Verlag, Berlin Heidelberg (2005)
5. Fowler, M.: Language Workbenches: The Killer-App for Domain Specific Languages?, http://www.martinfowler.com/articles/languageWorkbench.html (2005)
6. Frankel, D.S.: Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley Publishing, Inc., Indianapolis, Indiana (2003).
7. Gurov, V. S., Mazin, M. A., Narvsky, A. S., Shalyto, A. A.: Tools for Support of Automata-Based Programming. J. Programming and Computer Software, Vol. 33, No. 6, pp. 343–355 (2007), http://is.ifmo.ru/articles_en/_ProCom6_07GurovLO.pdf
8. Kelly, S., Tolvanen, J.-P.: Domain-Specific Modeling. IEEE Computer Society Publications, New Jersey (2008)
9. Meta Programming System, http://www.jetbrains.com/mps/
10. MOFLON project, http://www.moflon.org/
11. Muller, P.-A., Fleurey, F., Jézéquel, J.-M.: Weaving executability into object-oriented meta-languages. In: Briand, L., Williams, C. (Eds.) International Conference on Model Driven Engineering Languages and Systems (MoDELS), LNCS vol. 3713, pp. 264–278. Springer-Verlag, Berlin Heidelberg (2005)
12. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.2 (2009), http://www.uml.org
13. Paraschenko, D., Shalyto, A., Tsarev, F.: Modeling Technology for One Class of Multi-Agent Systems with Automata Based Programming. In: IEEE International Conference on Computational Intelligence for Measurement Systems and Applications, pp. 15-20. IEEE Xplore, La Coruna (2006)
14. Ward, M.: Language Oriented Programming. In: Software - Concepts and Tools, Springer Berlin / Heidelberg, Vol.15, No.4, pp. 147-161 (1994)