Model-based Java software development technology

1. Background and scientific context

Model-based software development is a way to overcome the increasing complexity of software products and their changeability [36]. It is based on dividing the software development into two separated processes: *domain engineering* and *application engineering*. Both include software development as a part. The first process provides software assets for the use in the second process. *Software assets* are the reusable resources used in application engineering. Examples of software assets include domain models, software architectures, design standards, communication protocols, code components and application generators. This facilitates software development by raising the conceptual level of application programming.

Java platform is one of the most used platforms for developing software in a wide range of application domains. It has large number of packages supporting the software development. Java beans are available for facilitating the application software development by introducing reusable software components. However, there is still a need for highly automated model-based Java software development tools.

The idea of model-based software development is not new. It has been around for almost 30 years. It has been widely used in the development of embedded (real time) systems and in mechatronics. Its most successful applications are in simulation software, there are well known specialized products, mainly in simulation domain like Simulink [3], Modelica [6] etc. One continuous effort in this field is pursued in NASA [1]. Aerospace applications, including software for the International Space Station (ISS), use model-based development extensively. NASA puts strict requirements on the model-based software development. As the authors say, *the production-quality program synthesis is the keystone for full-cycle model-based programming*. Without this capability, model-based programming is limited to being a prototyping tool whose utility ends after detailed design, when the production code is developed manually. We completely agree with this statement, and take it into account in the development of our methods.

Considerable amount of work is being done in improving the existing UML-based approaches with the aim of providing automated support to the software development [5] and language development [31]. This approach is also related to Model Driven Architecture (MDA) advocated by the OMG group [26] and to the development of domain specific languages (DSL), see [4, 19], because they all have the development of user friendly and automated problem solving tools as a goal. This approach includes the usage of UML-based models and metamodels. It concentrates either on the research of transformation rules for transforming an initial specification (a model) into another model or an executable code [37], or on the development of rules that represent the operational semantics [5] or even immediately perform the required computations. Despite its popularity and numerous research papers, this approach has remained rather theoretical. Sound criticism on this approach can be found in [22, 30]. As UML does not have a well defined semantics, then it is open to ambiguities and misunderstanding. In particular, the universal character of UML and its orientation at software implementation is an obstacle for its usage in domain modeling where the high-level domain specific concepts must be handled.

Another research direction in the model-based software development is the direct usage of graphical tools (without transformation of DSL or source models into the UML form) [32], e.g. the MetaEdit+, see [20, 33], but no universal tools for high-level domain modeling exist yet. One looses the rich collection of UML-based presentations in this case, but gets more freedom in developing the automation methods.

One more approach to model-driven software development has been made by the Eclipse community. Eclipse Modeling Project [12] that includes Eclipse Modeling Framework (EMF), Graphical Modeling Framework (GMF) and the Generative Modeling Tools (GMT) is a relatively new collection of technologies for building DSLs. EMF provides a basis for abstract syntax development (Encore model) which corresponds (is mapped) to a textual concrete syntax or a graphical concrete syntax. EMP includes various components, such as UML2, OCL, QVT, EMOF, XMI, etc. that enable users to develop DSLs using MDA. Generally speaking, EMP is a powerful tool, but it requires a lot of effort to develop a working DSL from scratch.

The important topic relevant to our project is a *generative programming* paradigm. It is about manufacturing software products out of components in an automated way [2]. This definition precisely fits into our model-based technology proposal. To go into more details, the generative programming focuses on software system families rather than one-of-a-kind systems. Such family members can be automatically generated based on a common *generative domain model* that includes *implementation components* and the *configuration knowledge* mapping between a specification and a finished system or component. A good source of information that addresses the issues and presents tools and applications of the generative programming is the GPCE international conference, see proceedings [7].

Our proposed project can be considered as a continuation of our earlier research on developing a visual tool [8, 9, 10] that can be used for specifying models. In this sense, it supports the direct usage of graphical tools, and does not require transformation of DSL or source models into the UML form. The project has similarities with the NASA approach, but relies on other program construction methods.

We have first positive experiences with applications of program synthesis in cyber defense and information assurance [23, 24, 25], in simulation [13, 14, 27], as well as in composition of services on large service models in e-government domain [15, 16, 17]. As a part of the proposed work, the developed software technology will be thoroughly tested in most of these application areas.

2. Goals and basic hypotheses of the proposal

The project is intended for application of new ideas in Java software development – a model-based software technology that is based on automatic code construction by means of new logical and planning methods. Proof-of-concept of these ideas exists in the form of a visual programing environment CoCoViLa [9].

The aim of the present project is 1) to develop a model-based Java software technology supported by automated program generation and respective methodologies, 2) to implement a tool for Java applications development based on this technology, and 3) to validate the technology through the development of industrial applications (e.g. fluid power systems and

technical chain systems, etc.) and public e-services. The tool is expected to be a software environment with strong visual part for developing and applying models as well as for controlling the computations. The developed model-based technology will provide high degree of automation based on application of logical methods of program construction. A workflow of the software development according to this technology is domain model driven as depicted in the following figure.

The workflow starts with domain engineering that provides the reusable core assets used during the process of application engineering for creating individual applications.
The domain engineering is the process that consists of the following three steps: domain analysis, design and implementation. Through the domain analysis domain experts learn users' requirements and collect information about the domain in order to create domain conceptualization and develop domain model. These form a basis for domain design that produces design patterns that are common for the applications within the domain. Domain implementation covers the implementation of components designed in the previous phase. According to the proposed technology, software developers create Java classes for domain components and their visual representation. In addition, on the basis of these reusable classes they develop meta-classes i.e. Java classes that are enriched with specifications for automatic program construction and visual components. In the application engineering phase, the application developers concentrate on requirements of a particular software product within the domain and write its specification in specification language of the tool of automatic program construction by reusing components of the domain model (i.e. meta-classes). The specification of the application is automatically transformed to corresponding valid logical representation for efficient generation of the customized program. The process of automatic program construction automatically generates the final executable Java code from the specification of the customized application.

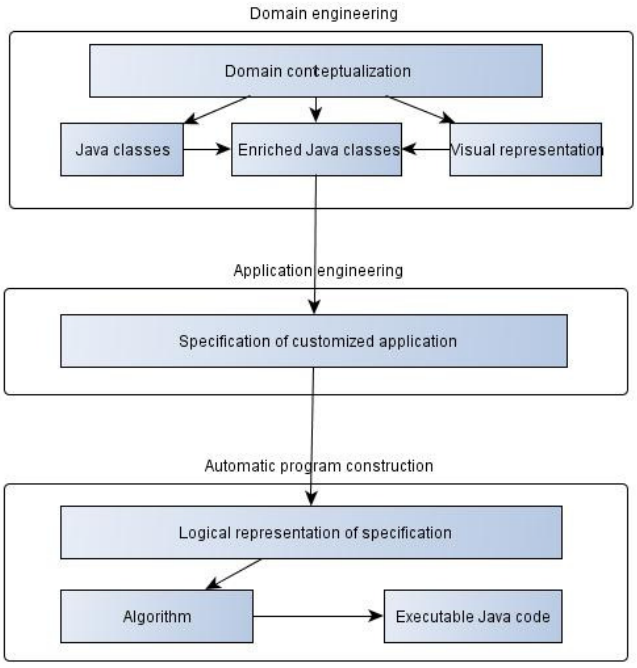

Figure. Proposed domain driven software development workflow

This workflow has been initially tested by our applications in simulation [13, 14].

The main hypothesis of our proposal is that in many cases (e.g. in simulation [13, 14]), instead of transforming specifications, first, from domain specific notations into UML, and thereafter using UML tools, as is the mainstream of model-based software development (and used in the model-driven architecture – MDA), one can transform the specifications directly into logic, and use logical tools for representation of semantics as well as for automatic construction of executable code. This hypothesis is supported by our experience in automated program construction [18] and, more generally, by transformation of high-level and visual specifications into executable code [11, 34]. In our case, UML can be used additionally for requirements specification and development of Java classes.
Our approach has a number of benefits and novel features as follows:

1. Reusability of components (software assets) developed during domain engineering phase;
2. Reduction of number of model-to-model transformation steps preceding code generation;
3. Reliable logic for program construction from specifications;
4. Reliable and reusable code for applications;
5. Separation of domain and application logic from operational logic and visual representation of application;
6. Specifications of different modeling levels (i.e. domain and application) are brought together in a consistent manner.

In addition, the project aims at achieving some theoretical results related to the logic of program construction as more expressive specifications are necessary for capturing conceptualizations of domains than used nowadays in the mainstream model-based software development. More expressive specifications require also more expressive logics to be used for automatic program construction. Therefore, first of all, we intend to extend the synthesis logic with means for representing resources that are available in the linear logic. We also consider Description Logics and ontology languages built on them (e.g. Ontology Web Language) as one of the options for formally representing domain conceptualizations (e.g. as related to e-services development domain).

3. Research methods and partners

The research performed in the project combines *application of logic* in development of algorithms with usage of *advanced software technology for implementation of visual and textual languages*. The goals of the project will be achieved by combining theoretical research focused on applying planning and algorithm synthesis methods (including the usage of linear logic for the synthesis of resource sensitive programs), with practical experiments in software development (performed in close cooperation with application developers).

The role of planning in automatic program construction being a starting point for extensions of our previous work within this project, has been described in detail in Chapter 4 of our book [28]. The usage of partial deduction for this purpose has been suggested by M. Matskin [18, 29] (The Royal Institute of Technology -- KTH, Stockholm) with whom we cooperate in the area of automatic program and e-service construction [15, 16, 17].

Higher-order attribute models and constructive logic will be used for implementing domain specific languages as a part of domain engineering. A theoretical basis and a concrete method for developing a universal visual tool and implementing domain specific languages is presented in our recent paper [11] that describes the translation of visual and textual

specification languages into higher order attribute models and equivalently – into the intuitionistic logic.

Cooperation with strong group of researchers in the field of software design, Kai Koskimies, Hannu Jaakkola and Jari Peltonen from Tampere University of Technology, is planned for software architectures design and model driven simulation related to our proposal.

Applications in simulation of fluid power systems are based on the original *multi-pole models* developed in the Tallinn University of Technology [13, 14]. We will continue the cooperation with this group of researchers and use the multi-pole modeling methods.

For e-services development in e-government domain, we propose semantic modeling of the domain by enriching specification of a domain by links to concepts of domain ontologies represented in Ontology Web Language (W3C standard). We have partially used this approach in our work on composition of semantic web services of e-government [16]. Co-operation in this field is continued with Estonian Information System's Authority and the Actors Ltd.

4. Expected impact of the project

The main impact will be a considerable improvement of software development processes in the area where model-based software technology is applicable: simulation, development of embedded software and e-services development. It is expected that the development of embedded software will be an important business area of small and medium enterprises (SME) in the IT field of Estonia, hence the project is essential for the innovation in IT SME. The technology, in particular, e-service composition, can be applied in large companies. It is especially suitable for companies that apply agile software development.

Availability of easy-to-use model-based software development tools will enable fast development of new simulation applications in the fields essential for Estonian public sector, economy and defense. Efficient development of new e-services for Estonian public sector will decrease government resources needed for their development and make e-services rapidly available for citizens.

The theoretical results of the project advance logical foundations of domain specification languages for automatic program construction from specifications. Using Description Logics combined with synthesis logic for formally representing domain conceptualizations will extend the usability domain of the methods of automatic program construction. Theoretical and experimental works in the simulation field have high impact to efficiency of industrial applications.

In conclusion, planned research results of the project will have great impact to the field of domain-driven software development in general, and to application engineering in the fields of simulation and development of e-services, in particular. The theoretical and experimental impact will be proven in a number of scientific papers that will be published in the relevant international journals (e.g. Computing and Informatics, Automated Software Engineering, Journal of visual languages and programming, etc.) as well as proceedings of the international conferences (e.g. MODELSWARD-International Conference on Model-Driven Engineering and Software Development, MS-Modeling and Simulation, EUROSIS, EGOVIS, JCKBSE, BalticDB&IS, etc.)

5. References

1. Cooke, D. E., Barry, M., Lowry, M., Green, C. NASA's Exploration Agenda and Capability Engineering. 2006.
2. Czarnecki, K., Eisenecker, U.W. Generative Programming: Methods, tools, applications, Boston, 2000.
3. Dabney, J. B., Harman, T. L. 1997 Mastering SIMULINK. Prentice Hall PTR.
4. van Deursen, A., Klint, P. Domain-specific language design requires feature descriptions. *Journal of Computing and Information Technology.* Volume 10, 2001.
5. Engels, G., Soltenborn, C., Wehrheim, H: Analysis of UML Activities Using Dynamic Meta Modeling. *In: Formal Methods for Open Object-Based Distributed Systems, 9th IFIP WG 6.1 International Conference*, FMOODS 2007, Paphos, Cyprus, June 6-8, 2007, pp. 76-90.
6. Fritzson, P. Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica. John Wiley & Sons, 2011.
7. Generative Programming and Component Engineering conference proceedings. http://www.informatik.uni-trier.de/~ley/db/conf/gpce/index.html.
8. Grigorenko, P., Saabas, A., Tyugu, E. Visual tool for generative programming. - *ACM SIGSOFT Software Engineering Notes*, 2005, 30, 5, 249-252.
9. Grigorenko, P., Saabas, A., Tyugu, E. COCOVILA – Compiler-Compiler for Visual Languages. *In: J. Boyland, G. Hedin. Fifth Workshop on Language Descriptions Tools and Applications LDTA2005.* ETAPS, 2005, p. 101 – 105.
10. Grigorenko, P., Tyugu, E. Deep Semantics of Visual Languages. In: E. Tyugu, T. Yamaguchi (eds.) *Knowledge-Based Software Engineering. Frontiers in Artificial Intelligence and Applications*, vol. 140. IOS Press, 2006, p. 83 - 95.
11. Grigorenko, P., Tyugu, E. Higher-Order Attribute Semantics of Flat Declarative Languages. *Computing and Informatics*, 2010 (in print).
12. Gronback, R. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley Professional, 2009.
13. Grossschmidt, G., Harf, M. Modelling and simulation of fluid power systems in an intelligent programming environment. *In: 6th International Industrial Simulation Conference 2008, ISC'2008* : June 9-11, 2008, Lyon, France, Proceedings: Ostend: EUROSIS, 2008, (A publication of EUROSIS-ETI), 224 - 230.
14. Grossschmidt, G., Harf, M. Multi-pole modelling and simulation of a hydraulic mechanical load-sensing system using the CoCoViLa programming environment. *In: Proceedings of 6th International Fluid Power Conference "Fluid Power in Motion", Dresden: 6th International Fluid Power Conference (6.IFK).* Dresden: Dresdner Verein zur Förderung der Fluidtechnik, 2008, 553 - 568.
15. Maigre, R., Grigorenko, P., Küngas, P., Tyugu, E. Stratified Composition of Web Services. *In: M. Virvou, T. Nakamura (eds.) Knowledge-Based Software Engineering. Proc. 8th JCKBSE.* IOS Press, 2008, p. 49 – 58.
16. Maigre, R., P. Grigorenko, H.-M. Haav, A. Kalja. 2012. A semantic method of automatic composition of e-government services. In *Databases and Information Systems VII: Selected Papers from 10th Int. Baltic Conf. on Databases and Information Systems, Baltic DB&IS 2012 (Vilnius, July 2012),* ed. A. Caplinskas, G. Dzemyda, A. Lupeikene, and O. Vasilecas, Frontiers of Artificial Intelligence and Applications, IOS Press, (to appear).
17. Maigre, R., Küngas, P., Matskin, M., Tyugu, E. Handling Large Web Services Models in a Federated Governmental Information System. *Proc. 3-rd International Conference on Internet and Web Applications and Services.* IEEE Computer Society & CPS, 2008, p. 626 – 631.
18. Matskin, M, Tyugu, E. Strategies of Structural Synthesis of Programs and Its Extensions. *Computing and Informatics. v.20*, 2001, p.1 -25.
19. Mernik, M., Heering, J., Sloane, A. When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)* Volume 37, Issue 4, 2005.
20. MetaCase Homepage. http://www.metacase.com.

21. Mints, G., Tyugu, E: Propositional Logic Programming and the Priz System. *J. Log. Program. 9 (2&3)*: 179-193, 1990.
22. de Niz, D. Diagram and Language for Model-Based Software Engineering of Embedded Systems: UML and AADL. Software Engineering Institute white paper, Carnegie Mellon University, 2007.
23. Kivimaa, J., Ojamaa, A., Tyugu, E. Graded Security Expert System. *Proc. CRITIS08*, Eds. R. Setola, S. Geretshuber. Berlin : Springer, 2009, 279-286. (Lecture Notes in Computer Science ; 5508).
24. Kivimaa, J., Ojamaa, A., Tyugu, E. Managing Evolving Security Situations. *MILCOM 2009: Unclassified Proceedings*, October 18-21, 2009, Boston, MA. Piscataway, NJ: IEEE, 2009, 1-7.
25. Kivimaa, J., Ojamaa, A., Tyugu, E. Pareto-Optimal Situation Analysis for Selection of Security Measures. *Proc. MilCom 2008*, 7 p.
26. Object Management Group: Model Driven Architecture. http://www.omg.org/mda.
27. Ojamaa, A., Tyugu, E. Rich Components of Extendable Simulation Platform. *Proc. WORLDKOMP'07: MSV2007, CSREA Press, 2007*, p. 121 - 127. M. Virvou, T. Nakamura (eds.) Knowledge-Based Software Engineering. Proc. 8th JCKBSE. IOS Press, 2008, p. 49 – 58.
28. Tyugu, E. Algorithms and Architectures of Artificial Intelligence. Frontiers in Artificial Intelligence and Apploicatons. IOS Press. 171 p, 2007.
29. Rao, J., Küngas, P., Matskin, M. Composition of Semantic Web services using Linear Logic theorem proving. *Inf. Syst. 31(4-5)*: 340-360, 2006.
30. Rath, I. Declarative Specification of Domain Specific Visual Languages. Master's thesis. Budapest, 2006.
31. Selic, B. A Systematic Approach to Domain-Specific Language Design Using UML. *In Proceedings of the 10th IEEE international Symposium on Object and Component-Oriented Real-Time Distributed Computing* (May 07 - 09, 2007). ISORC. IEEE Computer Society, Washington, DC, 2-9. 2007.
32. Sprinkle, J., Karsai, G. A domain-specific visual language for domain model evolution*, Journal of Visual Languages and Computing*, Vol 15 (3-4), Elsevier 2004.
33. Tolvanen, J.-P., Kelly, S. Metaedit+: defining and using integrated domain-specific modeling languages. *In S. Arora and G. T. Leavens, editors, OOPSLA Companion*, pages 819–820. ACM, 2009.
34. Tyugu, E., Valt, R. Visual programming in NUT. *Journal of visual languages and programming, v. 8*, pp. 523 – 544, 1997.
35. Veanes, M., Grigorenko, P., de Halleux, P., Tillmann, N. Symbolic Query Exploration. *In: ICFEM'09, Springer Verlag*, December 2009.
36. Vitkin, L., Dong, S., Searcy, R. and Manjunath, B. Effort Estimation in Model-Based Software Development. 2006 SAE World Congress Detroit, Michigan, 2006.
37. Whittle, J.: Transformations and software modeling languages: Automating transformations in UML. *In: J´ez´equel, J.-M., Hussmann, H., Cook, S. (eds.) Proc. Fifth International Conference on the Unified Modeling Language – The Language and its Applications, LNCS, vol. 2460*. Springer-Verlag, Dresden, Germany, 2002, pp. 227–242.