A Model-based Software Technology Proposal

Vahur Kotkas, Jaan Penjam, Ahto Kalja and Enn Tyugu
Institute of Cybernetics at Tallinn University of Technology, Akadeemia 21, Tallinn, 12618 Estonia

Keywords: Model-based Software Development, Application Engineering.

Abstract: In this paper we propose a technology for model-based software development. The technology separates domain engineering and application engineering and automates the actual executable code generation. This approach has been used extensively for simulations and we believe it is also applicable for more general software development.

1 INTRODUCTION

Model-based software development is a way to overcome the increasing complexity of software products and their changeability (Vitkin et al., 2006). 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.

2 MODEL-BASED SOFTWARE DEVELOPMENT APPROACHES

The idea of model-based software development is not new. It has been around for almost 30 years, but has not become a widely accepted paradigm. Its most successful applications are in simulation software, there are well known specialized products like Simulink (Dabney and Harman, 1997) or Scicos (Scicos Wiki). One continuous effort in this field is pursued in NASA (Cooke et al., 2006). 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 (Engels et al., 2007) and language development (Selic, 2007). This approach is also related to Model Driven Architecture (MDA) advocated by the OMG (Object Management Group) and to the development of domain specific languages (DSL), (see van Deursen and Klint, 2001; or Mernik et al., 2005), 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 (Whittle, 2002), or on the development of rules that represent the operational semantics (Engels et al., 2007) 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 (de Niz, 2007; Rath, 2006). In particular, the universal character of UML and its orientation at software

Kotkas V., Penjam J., Kalja A. and Tyugu E..
A Model-based Software Technology Proposal.
DOI: 10.5220/0004348203120315
In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2013), pages 312-315
ISBN: 978-989-8565-42-6

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) (Sprinkle and Karsai, 2004), e.g. the MetaEdit+ (see MetaCase, 2012; or Tolvanen and Kelly, 2009), but no universal tools for high-level domain modeling exist yet. One loses 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 (EMP) (Gronback, 2009) 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 technology is a *generative programming* paradigm. It is about manufacturing software products out of components in an automated way (Czarnecki and Eisenecker, 2000). 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 GPCE proceedings).

(Grigorenko et al., 2005-1; 2005-2; 2006) describes a visual tool 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 technology has similarities with the NASA approach, but relies on other program construction methods.

Positive experiences are recieved with applications of program synthesis in cyber defence and information assurance (Kivimaa et al., 2008-2009), in simulation (Grossschmidt and Harf, 2008, 2009; and Ojamaa, 2008), as well as in composition of services on large service models (Maigre, 2008).

3 GOALS AND BASIC HYPOTHESES OF THE PROPOSED APPROACH

The approach is intended for approbation of new ideas in software engineering – a model-based software technology that is based on automatic code construction by means of logical and planning methods.

The proposed 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 presented on Figure 1.

The domain engineering is the process where domain experts develop/obtain ontology (1) for a software system.

Software developers implement this ontology by developing classes (2), visual classes and metaclasses (4). Metaclasses are code components (classes are basic building blocks to be included in the developed software) extended with specifications that enable their automated handling during the steps (7), (8) and (9). Visual classes are metaclasses with visual representation to be used in step (6) for creating schemes.

The application engineering consists of requirements development (5) for a particular software product and of specification writing (6) in a precise language that is an input language of the automation tool.

The steps (7, 8, 9) from specification to final code are performed automatically (see Matskin and Tyugu, 2001 for an idea).

This is the scheme tested already (also in large-scale) for complex simulations (Grossschmidt 2008, 2009; Maigre, 2008; Harf and Grossschmidt, 2012).

The main hypothesis is that 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

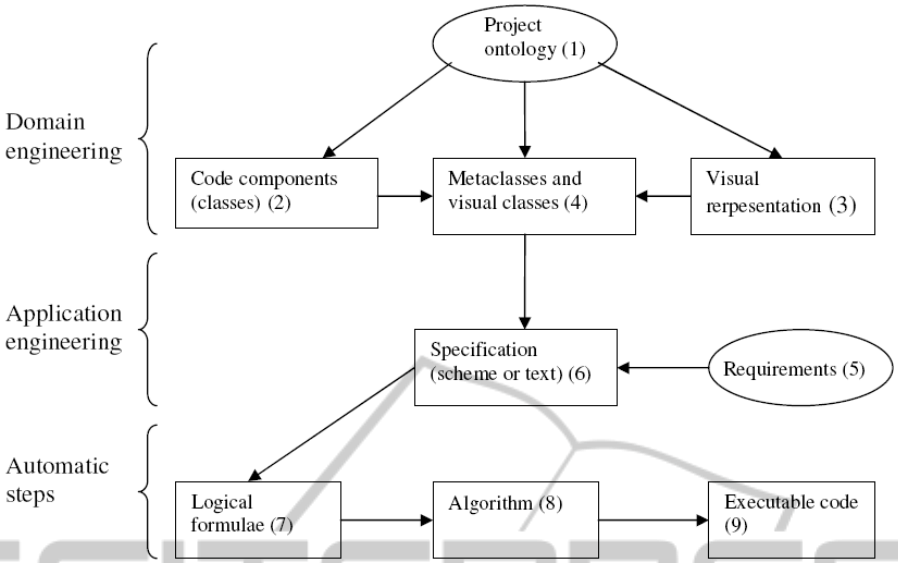

Figure 1: Software development workflow

tools for representation of semantics as well as for automatic construction of executable code. This hypothesis is supported by experience in automated program construction (Matskin and Tyugu, 2001) and, more generally, by transformation of high-level and visual specifications into executable code (Grigorenko and Tyugu, 2010; Tyugu and Valt, 1997). UML can be used in a conventional way for requirements specification and development of classes even in this case. However, the real success of the technology can be achieved when the logic of synthesis is extended and more expressive specifications, than used today in the model-based software development, will be available. We foresee the need for branching, resource consumption and distribution specification support, e.g. extension with linear-logic.

4 CONCLUSIONS

An approach of model-based software development technique is presented. Our belief is that such technique becomes applicable in practice only when automatic code construction by means of logical and planning methods is applied. For that the development has to be separated into several levels of engineering where domain engineering and application engineering is separated. The proposed approach needs more effort being put in domain engineering and application engineering while developing the executable code is fully automated.

ACKNOWLEDGEMENTS

This research was partially supported by the target-financed theme No. SF0140007s12 of the Ministry of Education and Research as well as by the ERDF funded Centre of Excellence in Computer Science.

REFERENCES

Vitkin, L., Dong, S., Searcy, R. and Manjunath, B., 2006. *Effort Estimation in Model-Based Software Development.* 2006 SAE World Congress Detroit, Michigan.

Dabney, J. B., Harman, T. L., 1997. *Mastering SIMULINK*. Prentice Hall PTR.

Scicos Wiki, http://en.wikipedia.org/wiki/Scicos

Cooke, D. E., Barry, M., Lowry, M., Green, C., 2006. *NASA's Exploration Agenda and Capability Engineering*.

Engels, G., Soltenborn, C., Wehrheim, H., 2007. *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.

Selic, B., 2007. 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.

Object Management Group. *Model Driven Architecture. http://www.omg.org/mda.*

van Deursen, A., Klint, P., 2001. *Domain-specific language design requires feature descriptions*. Journal of Computing and Information Technology. Vol. 10.

Mernik, M., Heering, J., Sloane, A., 2005. *When and how to develop domain-specific languages*. ACM Computing Surveys (CSUR) Volume 37, Issue 4.

Whittle, J., 2002. *Transformations and software modeling languages: Automating transformations in UML*. LNCS, vol. 2460. Springer-Verlag

de Niz, D., 2007. *Diagram and Language for Model-Based Software Engineering of Embedded Systems: UML and AADL.* Software Engineering Institute white paper, Carnegie Mellon University.

Rath, I., 2006. *Declarative Specification of Domain Specific Visual Languages. Master's thesis.* Budapest.

Sprinkle, J., Karsai, G., 2004. *A domain-specific visual language for domain model evolution.* Journal of Visual Languages and Computing, Vol 15 (3-4), Elsevier.

MetaCase, 2012. *http://www.metacase.com.*

Tolvanen, J.-P., Kelly, S., 2009. *Metaedit+: defining and using integrated domain-specific modeling languages.* OOPSLA 2009 Companion, pages 819–820. ACM.

Gronback, R., 2009. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley Professional.

GPCE proceedings. http://www.informatik.uni-trier.de/~ley/db/conf/gpce/index.html.

Grigorenko, P., Saabas, A., Tyugu, E., 2005. *Visual tool for generative programming.* ACM SIGSOFT Software Engineering Notes, 30, 5, 249-252.

Grigorenko, P., Saabas, A., Tyugu, E., 2005. *COCOVILA – Compiler-Compiler for Visual Languages*. In: J. Boyland, G. Hedin. Fifth Workshop on Language Descriptions Tools and Applications LDTA2005. ETAPS, p. 101 – 105.

Grigorenko, P., Tyugu, E., 2006. *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, p. 83 - 95.

Kivimaa, J., Ojamaa, A., Tyugu, E., 2009. *Graded Security Expert System.* Proc. CRITIS08, LNCS 5508 Springer, 279-286.

Kivimaa, J., Ojamaa, A., Tyugu, E., 2009. *Managing Evolving Security Situations. MILCOM 2009: Unclassified Proceedings, October 18-21, 2009, Boston, MA. Piscataway, NJ: IEEE, 1-7.*

Kivimaa, J., Ojamaa, A., Tyugu, E., 2008. *Pareto-Optimal Situation Analysis for Selection of Security Measures. Proc*. MilCom 2008, 7 p.

Grossschmidt, G., Harf, M., 2008. *Modelling and simulation of fluid power systems in an intelligent programming environment.* Proc ISC'2008 : June 9-11, 2008, Lyon, France, Proceedings: Ostend: EUROSIS, 2008, (A publication of EUROSIS-ETI), 224 - 230.

Grossschmidt, G.; Harf, M., 2009. *COCO-SIM - Object-oriented Multi-pole Modelling and Simulation Environment for Fluid Power Systems.* Part 1: Fundamentals. International Journal of Fluid Power, Vol. 10, No. 2, 2009, pp. 91 - 100. Part 2: Modelling and simulation of hydraulic-mechanical load-sensing system. International Journal of Fluid Power, Vol. 10, No. 3, 2009, pp. 71 - 85.

Harf, M.; Grossschmidt, G., 2012. *Modeling and simulation of an electro-hydraulic servovalve in an intelligent programming environment.* In: ASME 2012 11th Biennal Conference on Engineering Systems Design and Analysis (ESDA 2012), July 2-4, 2012, Nantes, France, [Proceedings]; New York, ASME, 2012, 1-9.

Ojamaa, A., Tyugu, E., 2008. *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, p. 49 – 58.

Maigre, R., Grigorenko, P., Küngas, P., Tyugu, E., 2008. Stratified Composition of Web Services. In: M. Virvou, T. Nakamura (eds.) Knowledge-Based Software Engineering. Proc. 8th JCKBSE. IOS Press, p. 49 – 58.

Maigre, R., Küngas, P., Matskin, M., Tyugu, E., 2008. *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, p. 626 – 631.

Matskin, M, Tyugu, E., 2001. *Strategies of Structural Synthesis of Programs and Its Extensions.* Computing and Informatics. v.20, p.1 -25.

Grigorenko, P., Tyugu, E., 2010. *Higher-Order Attribute Semantics of Flat Declarative Languages*. *Computing and Informatics.*

Tyugu, E., Valt, R., 1997. *Visual programming in NUT.* Journal of visual languages and programming, v. 8, pp. 523 – 544.

Czarnecki, K., Eisenecker, U. W., 2000. *Generative Programming: Methods, tools, applications*, Boston.