TOWARDS THE AUTOMATIC ARCHITECTURE DRIVEN SOFTWARE MODERNIZATION

Dimitar Birov and Yanka Todorova
Faculty of Mathematics and Informatics, Sofia University, James Bouchier blvd. 5, Sofia, Bulgaria
Dimitar.Birov@fmi.uni-sofia.bg, Yanka@Todorff.co.uk

Keywords: Software modernization, Model driven software development (MDD), Model driven architecture (MDA), Model transformation, Domain specific language (DSL), Business process modelling.

Abstract: In this paper, we describe a strategy for implementing source code analysis, model extraction, edition and analysis and code generation tools that can be applied to a software modernization of existing legacy software intensive system. As well we present an integrated approach focused on model driven architecture for software modernizations. Started with an extraction models from source code and other available software artefacts, transforming these models in order to obtain modern structure of software, and then generate a code from these models. Bussiness modeling tools and models naturally fits in the proposed constuction. A tools implementing domain specific languages are integrated into framework in vertical pipeline toolsuite.

1 INTRODUCTION

The constant evolution of software technology leads to continuous modernization of IT systems and software. Software modernization is a main driver of software evolution. Current enterprise IT systems are very complex, large and dispersed, which makes tasks of modernization non-attractive from business point of view. Moreover modernization of complex, software intensive systems is very expensive task. The reasons for software modernizations can be very different: from technology driven (the obsolescence of a technology) through the increasing users business needs (continuously changing user requirements) to market and business reasons (integration of enterprise IT systems in merging companies).

Full redesign and redevelopment of legacy system is not possible in most cases due to lost domain knowledge and technical skills for modernization. Model-driven software development (Stahl, Voelter, Czarnecki, 2006) offers an opportunity for increasing the automation in software modernizations. The full automation of this process could not be performed because of internal and external software quality attributes have to be established like maintainability, testability, reliability, security, etc.

This paper revisiting the possibilities to semi-automate the processes of IT systems modernization. We based on extraction models from source code and other available software artefacts, transforming these models in order to obtain modern structure of software, and generate a code from these models. Adding bussiness modeling tools and models produced of them and merging these models with others is one of the contribution of this paper. Different tools supporting described process of software modernization exists, but they lack of integration in one and the same software development framework or ingerated development environment. Second contribution of the paper is a common framework or toolsuite with vertical integration of tools. As well the toolsuite reflects 4 model levels of abstraction.

The paper is organized as follows: Sections 2 gives an overview of domain specific languages. Domain specific languages are used in different stages of software development. They automate some software engineers and design activities. Section 3 overview some bridges between technologivcal spaces. During software development software engineers use knowledge, tools and experience from different areas - technology spaces. Section 4 presents proposed approach based on the vertical pipeline scenario for possible software

Birov D. and Todorova Y.
TOWARDS THE AUTOMATIC ARCHITECTURE DRIVEN SOFTWARE MODERNIZATION.
DOI: 10.5220/0004459501360142
In *Proceedings of the First International Symposium on Business Modeling and Software Design (BMSD 2011)*, pages 136-142
ISBN: 978-989-8425-68-3

modernization focused on architecture driven approach of software development.

2 DOMAIN SPECIFIC LANGUAGES

A domain specific language (DSL) (Fowler, 2010) is a (programming, specification, modelling) language dedicated to a particular domain or problem. The advantage of a domain-specific language is that it provides appropriate built-in abstractions and notations according the problem domain. DSLs are used in a broad range of application domains-widely known example of DSL is MS Excel (Excel, 2010). Some software configuration or script languages can be viewed as domain specific. Software engineering and software architecture are very interesting problem domains concerning all process, stages and expertise of software development. In other words each DSL is specialised for a set of problems that share enough characteristics that it is worthwhile to study them as a whole. In this paper we focus on using DSL for software modernization. The problems of design and creating tools (compiler, interpreters) are out of scope of the paper and a lots of publications could be found.

Software modernization consists of following stages – first, extraction models from software artefacts (source code, databases, etc.), second extracted models are transformed to models, which are appropriate for modelling software components and software architecture, and third stage is automated code generation from obtained models. These three stages are supported with specific DSLs. The architecture driven software modernization is a specific problem domain with a knowledge, experience and technologies behind. This makes DSLs very useful instrument in the domain of modernization of software intensive systems.

DSL allows the different domain experts to be involved in the software development process. Domain experts can be software users who shares domain knowledge with the developers. It is not common software engineers to be an expert in the domain and additional resources have to be planned for their education. On the other hand, domain experts are often needed not only during the requirements specification, but also in the any stage of the development. These stages include design, modelling, verification and testing phases. If properly designed, DSLs provide a chance to involve these domain experts in the design of complex models like software architecture or IT architecture. DSLs extend the range and collaboration of people being able to contribute to the software modernization of the product. If the software developing team use the specific languages that each party is a familiar, this will decrease time for production and will increase a quality of software product and will shorten the time for production.

Software (architecture) models can be specified using DSLs. DSLs provide enough abstraction that they can serve as model specification language during the design phase of a software development. Because many technical details are already built in the semantics of the DSL, the specification written in a DSL can often be used to automatically generate code that forms the implementation. Thus, DSLs often bridge the gap that exists between the phases of the software engineering process, especially between the design and the implementation phase. As we see later software designers can use DSLs for model transformation which allows in pipeline manner to transform one model to another, more abstract or more concrete model depending on purpose. Pipeline transformation allows not losing knowledge and expertise during transformation and same time increase an abstract level of models. As well each transformation step can be checked syntactically and semantically.

The DSL is good for documentation purposes in order to ease the communication between developers and customers due to semantics included in DSL. If the semantics of the DSL is formally specified using some mathematical notation, then the DSL can be used as specification language also, because an unambiguous description of the semantics exists.

2.1 Business Process Modelling

Business processes (Boev, Surova, Nikolov, Zhivkov, 2011) mirror business activities in the company. BPMN (Shapiro, White, Palmer, 2011) is a notation for graphical presentation of business processes and model business activities according to domain experts and business analysers. The knowledge encoded in diagrams will be saved during model transformation on later stages with purposes of optimization of business process. We could not generate executable code directly from these diagrams but after appropriate transformation we can reach models suitable for software assets (source code, database schema, configuration script, etc.) generation. After modification of the model

modified source code can be obtained semi automatically, which is syntax correct with respect of some general purpose language (GPL).

2.2 DSL Categories

Two types DSL are known – textual and graphical domain specific languages. Textual DSLs can be easily embedded in other general purpose languages (GPL). The editors for embedded languages are widely used and users are familiar with it. Graphical DSLs are more intuitive for domain experts and can be embedded in some graphical language like UML or BPMN language. Textual DSLs can be less understandable for domain experts, while for graphical DSLs can be difficult to develop tools with appropriate quality.

Two other categories of DSLs are internal (embedded) and external DSLs. Internal DSL is implemented inside of general purpose host language, and their characteristics vary depending on the features of base language. Sometimes embedded DSL is implemented as a library or framework. External DSL gives a maximum syntax and expressions freedom and requires a good language development support. DSLs discussed in the rest of the paper are external. This means that they have notation used by domain experts.

3 BRIDGING TECHNOLOGY SPACES

Model-driven approaches move focus of software modernization from last generation programming languages code to models expressed in some modelling language - UML for example. Models can graphically depict system's structure and behaviour at a certain point of abstraction. We can refer to a source code as a textual representation of a model of design concepts. In this paper this understanding is important, because we are not focused on the source code analysis and transformation techniques and approaches in details but we are going to treat source code as input for a model extraction and as an output after code generation from a model of design concepts.

3.1 Technological Spaces

The term technological spaces (TS) was initially proposed by Kurtev et al. (Kurtev, Bezivin, Aksit, 2002) to name "a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities" It is often associated to a given user community with shared know-how, educational support, and common literature. It is also special network of exchange expertise and ongoing research and a repository for abstract and concrete resources.

Five technological spaces are commonly recognized: Programming languages concrete and abstract syntax, Ontology engineering, XML-based languages, Data Base Management Systems (DBMS), Model-Driven Architecture (MDA). Each technology space is defined according to a couple of basic concepts: Program/Grammar for the Syntax TS, Document/Schema for the XML TS, Model/Meta-Model for the MDA TS, Ontology/Top-Level Ontology for the Ontology engineering TS and Data/Schema for the DBMS TS.

In this paper we outline the bridges between (abstract) Syntax TS (known as *grammarware*), XML TS (known as *documentware*) and Model TS (known as *modelware*) and integration of bodies of knowledge developed by different research software development communities. The grammarware technological space is concerned with grammars, grammar-based description languages, and associated tools. The modelware technological space is concerned with metamodels, model-based description languages, and associated tools, documentware is concerned with XML, XSLT and associated tools. The bridges between these three TS establish foundation of integration between transformation tools discussed in the paper. Integration is based on a physical, a logical, and a pragmatical bridge between grammarware language and modelling framework.

3.2 Bridge Grammarware to Modelware

Most of modernization scenarios (Ulrich, Newcomb, 2010) involve dealing with source code conforming to the grammar of a programming language. Some of additional software assets (like configuration files, script files, resource description files) which are formed using a formal language can be analyzed or manipulated by tools. These tools can automate knowledge extraction from software assets in some degree. This way the tools are significantly support bridging and understanding knowledge between the grammarware and modelware TS. Extracting and creating models from source code is a first step for architecture/model driven software modernization.

Two main groups of bridging approaches are known - approaches focused on grammars or syntax oriented approaches and approaches focused on models or model oriented approaches. Grammar-based approaches consider generation of metamodels from grammars, while (meta)model based approaches generate grammars from metamodel. Software modernization process starts with collection of some information from source code and other available software assets (user interfaces, databases, design documentation, configuration files, etc.) that's why grammar based approach prevails. Model-based approach suits significantly for code-generation phase in automated software modernization process, when from models a new code was generated which confirms a (previously specified) language grammar. Both process transforms text to model (T2M) and then model to text (M2T) in order to implement software modernization process.

The xText (Behrens, Clay, Efftinge, 2010) and the works of Wimmer et al. (Wimmer, Kramler, 2005) and (Kunert, 2006) are examples of grammar-based approaches. Operational semantics of modelled languages in modelware TS can be described formally as well (Sadilek, Wachsmuth, 2009).

This approaches lack of quality of generated models. M2M transformations are not easy and lots of manual work is needed to be performed in respect to obtain clean models suitable for code generation. These transformation languages do not provide construct to make transformation process easy. In order to obtain knowledge about the software system parsing of additional software artifacts has to be done. MoDisco (MoDisco, 2011) extract knowledge from different software artifacts during the model discovery phase - obtaining a model that represents a view on the legacy system (or at least parts of it) from its source code, raw data, available documentation, etc.. Next MoDisco phase consists of models analysis, particular model transformations are performed until the final (desired) software artifacts are obtained. MoDisco is an Eclipse open source project and is based on Eclipse Modelling Framework and integrates OMG/ADM standards (KDM, SMM).

Gra2Mol (Izquierdo, Cuadrado, Molina, 2008) is domain-specific model transformation language specially intended to deal with source code described by a grammar. Gra2MoL is a rule-based transformation language whose rules have a similar nature to that of other model transformation languages.

Each transformation definition consists of a set of transformation rules which specify relationships between grammar elements and metamodel elements. Gra2Mol raises significantly levels of abstraction of model extracted from source code. For example it is easy to extract knowledge KDM model from Java code.

3.3 Bridge Modelware to Grammarware

Widely used techniques in software development is (partially) code generation from models. The possibility to automate code generation process and obtain code straight from models adds to the flexibility, maintainability, and portability of application. The different tools exist depending on level of automation of the code generation process. In the next part of this section we will outline the WebDSL – DSL created for purposes of web applications. WebDSL (Hemel, Kats, Visser, 2008), (Hemel, Kats, Groenewegen, Visser, 2010), (Hemel, Groenewegen, Kat, Visser, 2011) allows to reduce amount of code developers need to write by introducing abstractions same time entire application is typechecked for errors.

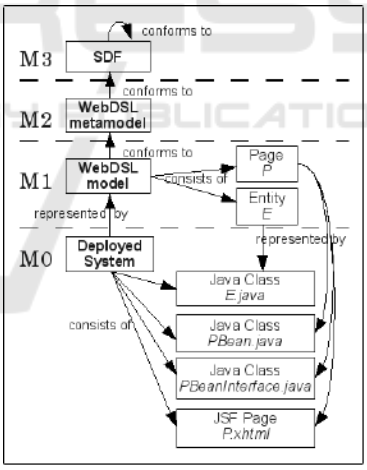

Figure 1: Organization of models of the WebDSL generator.

The architecture of WebDSL generator comprises the approach of code generation by model transformation, and follows the four-level model organization of Bezivin (Bezivin, 2005). Fig. 1. shows the model hierarchy – at top level (M3 meta-meta-model) is the grammar of the Syntax Definition Formalism SDF (Visser, 1997), (Heering, Klint, Rekers, 1990). SDF is intended for the high-level description of grammars for wide spectrum

computer-based formal languages: general purpose programming languages, domain-specific languages, data formats and others. Any SDF definition describes the syntax of the language and the following step is to generate a working parser from this definition.

The grammar of WebDSL is defined in SDF at the M2 level meta-model and describes the valid sentences of the language. From the grammar, the parser automatically can be generated. Generated parser transforms the textual representation of a model to an abstract syntax tree (AST). All subsequent transformations are applied to the AST corresponding to the textual representation of the model.

The WebDSL generator transforms high-level models into Java code and XML files. The bridge between three TS – model TS, syntax TS (Java) and document TS (XML) is hardly coded in functionality of the WebDSL. On three of them we could apply transformation in syntax TS. We could transform from one language to another use code-to-code (C2C) transformation. In document TS we could apply different schema for XML transformation. Bridging between syntax TS and document TS is a subject of study during the last decade and it is well understood and established.

The transformations (of WebDSL model) are expressed in Stratego (Bravenboer, M., Kalleberg, K. T., Vermaas, R., Visser, E.. 2008),(Visser, 2004) transformation language. Stratego/XT is a high-level term rewriting system which implements the paradigm of rewrite rules with programmable rewriting strategies and integrates M2M, model-to-code (M2C), and code-to-code (C2C) transformations. A strategy is essentially function that controls the order of application of more basic transformations. As well Stratego provides programmable strategies for building complex transformations that control the application of rules. In Stratego, the application of rewrite rules is under the control of programmable strategies, such that transformations can be explicitly staged.

Using strategies, the WebDSL generator is divided into different transformation stages. Actually the generator is organized as a pipeline of model-to-model transformations. Each stage consists of a set of rewrite rules that rewrite extensions of the WebDSL core language to more primitive language constructs. This technique of compilation by normalization has advantage to reduce the semantic gap between input and output model, this way avoiding the complexity associated by directly generating code from the input mode.

Model level (M1, Fig 1) of WebDSL models web applications, which consisting of entity and page definitions. At this level not all models that conform to the WebDSL syntax are valid. That is why semantic analysis needs to be performed. A separate type checking stage of the generator performs checks. If static semantic constraints are violated an error reported. The semantic information gathered at this stage is also used to provide context information for other transformations.

Level M0 presents the actual web applications consisting of programming language constructs (like Java classes) and web (XHTML) pages. These software assets represent the models at the M1 level. M0 models can be implemented in different languages like PHP, Python, JScript etc. This is very useful and important for next modernization and software evolution. For example product lines and mobile application can be developed with a small amount of efforts. Moreover that M0 systems can consists of high-level application frameworks, in case of Java these are Java Persistence API (JPA), JavaServer Faces (JSF), etc. In some other cases more elegant and flexible approach for implementation is to insert middle level of intermediate embedded DSLs into the general purpose implementation language.

Extensions of the WebDSL language, such as the access control and workflow abstractions are realized as plug-ins to the base language, extending the generator with new normalization rules.

4 PIPELINED TOOLSUITE

In this section we present our proposal for software modernization based on previously described two bridges. First of it based on a source code (P), available artefacts, and documentation Artefacts(P) = P + DataBasesSchema (used in P) + UserInterface (HTML, Asp, Jscript, PHP web pages, dialog boxes, XML files, etc.) + DocumentBase(about design of P). Part of the artefacts confirms to some set of grammars GrSet(Art(P)). From this set we can generate models of representation MRep(Art(P))

$$FMEstr: GrSet(Art(P)) \rightarrow MRep(Art(P)) \quad (1)$$

Model Extraction Functions (FMEstr) is a set of functions which extract models from grammars for each artefact of software application and source code.

MRep(Art(P)) is a set of models. Each model confirms to model from a set of metamodels MMRep(Art(P)). These metamodels represent a

basic knowledge about the source code and other artifacts of the existing system. Part of this metamodels could be KDM of OMG group, which is basic knowledge repository. Another metamodels could be Abstract and/or Concrete Syntax Trees of the source P. Another example could be entity relationships schema for databases. This part is very similar to this one implemented in Gra2Mol.

If we look back on models hierarchy at Fig.1., M0 level of model abstraction is a source code P and Art(P), M1 level is presented by MRep(Art(P)) and M2 level is represented by MMRep(Art(P)). M3 could be SDF for description of metamodel construction.

Additionally to this some business process specification can be obtained – if it is available we can use it as an artefact of software. So this way BPMN notation is a part of Art(P)). If it not exist based on the users interviews we could create it using visual modelling tools. Visual languages for BP Modelling are part of Syntax TS supporting BPM. As well visual languages are DSL languages with appropriate tools supporting modelling. As a DSL language they have a syntax (graphical symbols) and semantic (incorporated into the models). Many (visual) languages for modelling web applications have been developed. WebML (Brambilla, Comai, Matera, 2007) support generation of (web) application from BP specifications (Brambilla, 2006). Transformation from existing BPMN to WebML can be automated using DSL. As it is shown on Fig 2. through manual activity we can create Choreography model. Then through model transformation we can obtain refinement set of models.

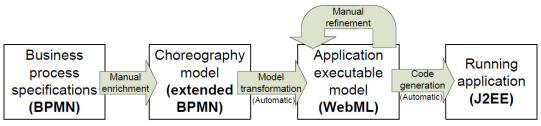

Figure 2: Automatic transformation of BPMN to WebML.

This way application executable model became a part of MRep(Art(P)) and respectively we can obtain extended part of MMRep(Art(P)). Merging models obtained both ways will enrich knowledge obtained from statically structured models with a knowledge of processes or dynamical models. Call Flow Graph (CFG) which is obtained from source code extraction as a part of MRep(Art(P)) carry some dynamical information but using BPMN models we could obtain almost complete information about dynamics of the process of execution.

The process of model transformation can be automated also. Using toolsuite like XTEAM-2 (XTEAM), (Edwards, Brun, Medvidovic, 2010) (eXtensible Tool-chain for Evaluation of Architectural Models) we can automate process of evaluation/creation of DSLs for manipulation of models. This type of tools performs model checking and model transformation. These tools operates on metamodel level M3 and transformed models to models through generated DSL.

$$\text{MMTools: MMRep(Art(P)) -> MMRep(Art(Q))} \quad (2)$$

Art(Q) is a new set of models of artifacts of a new modernized system Q. So next step from our proposal goes down from refined and appropriate metamodels to obtain concrete models (one of them is software architecture model) which will represent concrete artefacts of the new system Q. We got idea for this from process of code generation, described for WebDSL. We need particular DSLs for obtaining models for each one of the metamodel in MMRep(Art(Q)). Automatic creation of DSLs could be done with XTEAM.

Code generation step is very trivial and well-studied – from models of artefacts of Q we can create the concrete exemplars for concrete platform. This approach makes process of software modernization very flexible.

5 CONCLUSIONS

In this paper we present an integrated approach focused on model driven architecture for software modernizations. Started with an extraction models from source code and other available software artefacts, transforming these models in order to obtain modern structure of software, and generate a code from these models. Bussiness modeling tools and models naturally fits in the proposed constuction. A recent (versions of) tools from different technological spaces are integrated into framework in vertical pipeline toolsuite.

Some details of integration can be a subjects for future research. The protoype of this framefork in Eclipse integrated development environnment is under development.

ACKNOWLEDGEMENTS

This research was fully supported by the Bulgarian National Science Fund under Grant DO 02-102/23.04.2009.

REFERENCES

Behrens X., Clay M., Efftinge, C.,et al. *Xtext User Guide*, http://www.eclipse.org/Xtext/documentation/1_0_1/xtext.pdf, 2010

Bezivin, J. *On the unification power of models*. Software and System Modeling, 4(2):171–188, 2005.

Boev, S., Surova, E., Nikolov, K., Zhivkov, V., *Incorporating collaboration In Business Processes*., First International Symposium on Business Modeling and Software Design, BMSD 2011

Brambilla, M., *Generation of WebML Web Application Models from Business Process Specifications*, Demo at 6th International Conference on Web Engineering (ICWE2006), July 2006, Palo Alto, CA, USA

Brambilla, P. F. M., Comai S., Matera, M.. Designing web applications with WebML andWebRatio. In G. Rossi et al., editors, Web Engineering: Modelling and Implementing Web Applications, Human-Computer Interaction Series. Springer, October 2007.

Bravenboer, M., Kalleberg, K. T., Vermaas, R., Visser, E.. *Stratego/XT 0.17. A language and toolset for program transformation*. Science of Computer Programming, 72(1-2):52-70, 2008.

Edwards, G., Brun, Y., Medvidovic, N., *Automated Analysis and Code Generation for Domain-Specific Models*, Technical Report USC-CSSE-2010-517, Center for Software and Systems Engineering, University of Southern California, August 2010.

Excel, http://office.microsoft.com/en-us/excel/

Fowler M., *Domain-Specific Languages*, book, Addison-Wesley, 2010, ISBN: 0321712943, 9780321712943

Heering J., Klint P., Rekers, J. *Incremental generation of parsers*. 1344--1350. IEEE Transactions on Software Engineering. 16. 12. 1990.

Hemel Z., Kats L. C. L., Visser E. *Code Generation by Model Transformation. A Case Study in Transformation Modularity*. ICMT 2008, LNCS 5063, pp 183—198. Springer, June 2008. (An updated, extended version was published in 2009 in SoSyM.)

Hemel Z., Kats L. C. L., Groenewegen, D., Visser, E. *Code Generation by Model Transformation. A Case Study in Transformation Modularity*. Software and Systems Modeling, Vol. 9, Iss. 3, pp 375—402, Springer, 2010.

Hemel, Z. , Groenewegen D. M., Kats L. C. L., Visser, E.. *Static Consistency Checking of Web Applications with WebDSL*. Journal of Symbolic Computation, Volume 46, Issue 2, Elsevier, 2011.

Izquierdo, J. L. C., Cuadrado, J. S., Molina, J. G.. *Gra2MoL: A domain specific transformation language for bridging grammarware to modelware in software modernization*. In Workshop on Model-Driven Software Evolution, 2008

Kunert, A., *Semi-automatic generation of metamodels and models from grammars and programs*, Fifth Intl. Workshop on Graph Transformation and Visual Modeling Techniques, E. N. in Theorical Computer Science, Ed., 2006.

Kurtev I., Bezivin, J., and Aksit M.. *Technological spaces: An initial appraisal*. In CoopIS, DOA'2002 Federated Conferences, Industrial track, 2002

MoDisco, http://www.eclipse.org/MoDisco/

Sadilek, D., Wachsmuth, G., *Using Grammarware Languages To Define Operational Semantics of Modelled Languages*, TOOLS '09: 47th International Conference Objects, Models, Components, Patterns, Springer, 2009.

Shapiro, R., White, St., Palmer, N., et al, BPMN 2.0 Handbook, book, 2011, ISBN: 9780981987033

Stahl, T., Voelter, M., Czarnecki K., Model-Driven Software Development: Technology, Engineering, Management, book, John Wiley & Sons, 2006 ISBN:0470025700

Visser, E. *Syntax Definition for Language Prototyping*. PhD thesis, University of Amsterdam, September 1997.

Visser, E. *Program transformation with Stratego/XT: Rules, strategies, tools, and systems in StrategoXT-0.9*. In C. Lengauer et al., editors, Domain-Specific Program Generation, LNCS 3016, pp 216–238. Spinger, June 2004.

Wimmer, M., Kramler, G.: *Bridging Grammarware and Modelware*. 4th Workshop in Software Model Engineering (WiSME'05), October 3rd, 2005.

Ulrich W, Newcomb Ph., *Information Systems Transformation: Architecture-Driven Modernization Case Studies*, Morgan Kaufmann OMG Press, 2010, ISBN: 0123749131

XTEAM http://softarch.usc.edu/~gedwards/xteam.html