Model-Based Code Generation

- PhD Thesis Summary -

Scientific Supervisor:
Prof. Dr. Bazil Pârv

Author:
Paul Horaţiu Stan

February 2012
## Contents

1 Introduction 4

2 Model Driven Architecture 11
   2.1 Overview .................................................. 11
   2.2 Model Transformation ...................................... 14
      2.2.1 Model To Text Transformation ......................... 15
      2.2.2 Text To Model Transformation ......................... 16
   2.3 Standards used by the MDA: UML, XMI and MOF ............... 17
   2.4 Model Driven Development Approach ........................ 18
   2.5 Software Component Architecture .......................... 20

3 Model Driven Architecture tools and frameworks 22
   3.1 Academic MDA Tools ....................................... 22
      3.1.1 Atlas Transformation Language ....................... 22
      3.1.2 Kermeta ................................................ 23
   3.2 Industrial MDA tools ....................................... 23
      3.2.1 Eclipse Modeling Framework ........................... 23
      3.2.2 Apache Velocity ....................................... 23
      3.2.3 Java Emitter Templates ................................. 24
      3.2.4 Acceleo ................................................ 24
      3.2.5 Xtext and Xpand ...................................... 24
      3.2.6 WebML and WebDSL ..................................... 25

4 An approach for platform interoperability based on proxy objects 27
   4.1 The Problem ................................................ 28
      4.1.1 Rationale .............................................. 28
      4.1.2 Current interoperability approaches .................... 28
      4.1.3 Drawbacks of current interoperability approaches .... 29
   4.2 Proposed Solution ......................................... 30
      4.2.1 Conceptual view of the proposed solution .............. 30
      4.2.2 Architecture .......................................... 31
      4.2.3 How it works .......................................... 32
   4.3 The Indep communication framework .......................... 36
      4.3.1 The Indep architecture ................................ 36
      4.3.2 How it works .......................................... 39
      4.3.3 Case studies .......................................... 46
   4.4 Code Generation Metrics .................................... 48
      4.4.1 The percentage of generated code lines ............... 48
      4.4.2 Percentage of generated classes ....................... 48
   4.5 Conclusions and Future Work ............................... 49
5 A solution for component-based developing using automatic code generation for component connections

5.1 The Problem

5.1.1 Rationale

5.1.2 Current component-based composition approaches

5.1.3 Drawbacks of the current component-based composition approaches

5.2 Proposed solution

5.2.1 Conceptual view

5.2.2 Glossary

5.2.3 Architecture

5.2.4 How it works

5.3 The Building Blocks Dev Studio component-based development framework

5.3.1 The Building Blocks Dev Studio architecture

5.3.2 Component communication

5.3.3 A CBD process based on Building Blocks Dev Studio

5.3.4 Case study: The Library Application

5.4 Code generation metrics

5.4.1 The percentage of generated code lines

5.4.2 The percentage of generated classes

5.5 Conclusions and Future Work

6 A DSL for the development of data-intensive applications

6.1 The Problem

6.1.1 Rationale

6.1.2 Drawbacks of the current DSLs for data intensive applications

6.2 The Proposed Solution

6.2.1 Conceptual view

6.2.2 Technical details

6.2.3 How it works

6.2.4 A comparison with WebML and WebDSL

6.3 The DevDSL plugin

6.3.1 The DevDSL architecture

6.3.2 How it works

6.4 Case studies

6.4.1 Case study 1: The Library Application

6.4.2 Case study 2: A transformation engine to PHP Code Igniter framework

6.4.3 Case study 3: An Order Processing System developed using DevDSL

6.4.4 Case study 4: A custom validation mechanism for DevDSL

6.5 Conclusions and Future Work

7 Conclusions

Appendix:

A The serialized model of the HelloWorld component

B The Library Application specification written in DevDSL language.
List of publications

Published papers (journals)


Published papers (proceedings)


Submitted papers


List of keywords

Model Driven Engineering, Model Driven Architecture, Domain Specific Modeling, Domain Specific Languages, Automatic Code Generation, Component-based development, Platform Independent Model, Platform Specific Model, Model Transformations, General Programming Languages.
Summary

This Ph.D. thesis is the result of my research in the field of Model Driven Engineering (MDE), particularly in Model Based Code Generation (MBCG), research which was started in 2008 under the supervision of Prof. Dr. Bazil Pǎrv.

Motivation

A simple and easy question is: Why the thesis title is Model Based Code Generation? The answers are:

- **MBCG** can be viewed as a subsection of MDE focused on model transformation engines. The source metamodel is the modeling language used to specify the platform-independent model (PIM) of a system and the target metamodel is the syntax of the programming language for which the engine will generate the source code, representing the platform-specific model (PSM) of the system.

- **MBCG** can improve the software development time due to the automated source code generation.

- **MBCG** can increase the software quality because the coding errors are eliminated.

- **MBCG** is an open research space, there are many articles, books and conferences focused on this domain.

- **MBCG** is the next step in software engineering and compilation. At the beginnings the human resources were forced to understand and to write programs in the computer languages, the second step introduced the assembly language which assigned a suggestive word to each computer instruction, at the third step programming languages and compilers were introduced, based on the language syntax the compiler checks a program and generates computer statements. MBCG can be viewed as the next step on this natural way, the models specified in a platform independent way using domain specific languages (DSLs) are checked and translated into general programming language (GPL) source code files.

- Using **MBCG** it is possible to reduce the gap between technical people and customers. A domain specific language oriented on the client problem can be designed and used in order to collect client requirements in a formal and non ambiguous way.
Research objectives

In the context on current open research problems in the field of MDE, the research objectives of this thesis were focused on automatic code generation based on application models. The following enumeration presents the main objectives:

- A proposed architecture for reusing PSMs in another platforms than the platforms in which has been designed to be used. The proposed approach uses a generated proxy object [108] which acts as a mediator between the new PSM and the old one. Each proxy object delegates the method calls to the old PSM.

- The proposal of a component-based development process which automate the source code generation. There are two aspects: (1) the generation of the skeleton source code of a component, not containing the business logic of a component, and (2) the generation of the source code for connecting components [102, 103, 104].

- The proposal of a DSL for data intensive applications which allows the automatic translation to different PSM, web oriented or not [107, 106, 105].

Each objective is described in detail in a dedicated chapter in a common way, first it is presented the problem, the second step describes the proposed solution, a case study or a list of case studies is presented at the third step and finally conclusions and future improvements or open problems are described.

Chapter 2 presents an overview about Model Driven Architecture [4, 116, 8, 87, 59, 114, 90, 22, 12, 91, 27, 33, 70]. The Model Drive Architecture targets are [116]: Technology obsolescence, Portability, Productivity and time-to-market, Quality, Integration, Maintenance, Testing and simulation and Return on investment. This chapter enumerates the major MDA concepts [4] such as: System, Model, Model driven, Architecture, Viewpoint, MDA viewpoints, Platform, Platform Independence, Platform Model, Model Transformation, Implementation, Computation Independent Model (CIM), Platform Independent Model (PIM) [26] and Platform Specific Model (PSM). OMG provided a general specifications, and as answers, MDA tools has been developed by research teams. There are many types of MDA tools, such as: [116]: Creation Tool, Analysis Tool, Transformation Tool, Composition Tool, Test Tool, Simulation Tool, Metadata Management Tool and Reverse Engineering Tool.

The basic MDA flow consists in defining a platform independent model (PIM) [26] and transforming it automatically to one or more platform-specific models (PSMs) [28]. Model transformation domain can be divided into [28]: (1) Model to text transformations and (2) Text to model transformations. At the end of this chapter a list of standards used by the MDA is presented, such as: XML, XMI, UML, MOF, MDDA, SCA, BPMN [8, 38, 23, 97, 125].
Chapter 3 presents a set of MDA tools and frameworks used in the research presented in this thesis. The presentation is structured into two main sections: (1) Academic Tools and (2) Industrial Tools. Academic MDA Tools section contains instruments developed by research teams from different universities and Industrial MDA Tools sections shows productions IDE’s and frameworks that are used by software development companies.

Personal contributions are presented into Chapters 4, 5 and 6.

- Chapter 4 "An approach for platform interoperability based on proxy objects": presents a technique for platform interoperability based on model transformation approaches. Instead of serializing objects for implementing the communication between platforms, the proposed approach involves generating proxy objects for the remote objects. The proxy objects types are generated automatically. The source code written for implementing the remote object functionality acts as a model that conforms to a metamodel (the programming language syntax), the proxy object’s type acts as a model that conforms to another metamodel (the syntax of the programming language used for implementing the proxy object). As a conclusion, the technique presented in this chapter can be integrated in the model transformation area contributions. The original solution is described in greater detail in [108].

- Chapter 5 "A solution for component-based developing using automatic code generation for component connections" proposes a technique for component-based software development. The novelty of the proposed solution resides in each component has public properties that can have one of the following directions: IN, OUT and INOUT, and the communication between components is implemented using these properties, called pins. As a result of the proposed technique, the components are not dependent on each another. The components are only dependent on their pins data types. The original contributions are presented in more detail in the papers [104, 102, 103].

- Chapter 6: "A domain-specific language for the development of data-intensive applications" presents a solution which involves (1) a language for defining the platform-independent model of an application and (2) a transformation engine which support translation from the platform-independent model to .NET web application model. Both packages are included into an Eclipse plug-in with code completion and syntax highlight. The original contributions from this chapter are described in more detail in the papers [107, 105, 106].

Chapter 4 presents a new approach regarding interoperability. The novelty of the proposed solution resides in using a proxy object for each used remote object in order to avoid serialization.

An actual software engineering problem is the improvement of the software development process and the final product quality [43]. In order to achieve this, the source code
should be reusable [73]. Obviously, it is good news for a developer if an old library developed in an old programming language can be used in a new programming language without being necessary to rewrite the code. In this case, it saves time and improves the quality of the final software system, because the old library has been tested in the past and it works fine.

In the context of model transformation, one target of this PhD thesis was to determine an algorithm for generating the source code for a specific programming language based on a source code written for another programming language in an automatically mode. This is not a translation of the source code from a programming language to another, the autogenerated code acts as a proxy to the real code, so it delegates the execution to the real routines. This research can be included into "Model Transformation" area because it transforms an executable model resulted from compiling a source code written for a programming language to a source code written in another programming language which acts as a proxy to the initial executable model.

There are many frameworks written for both Java and .NET, for instance: Hibernate [25] respectively NHibernate, JUnit [80] respectively NUnit etc. These frameworks could be written for a platform and reused from another platform, for instance instead, of rewriting Hibernate for .NET it can be reused directly. Chapter 5 uses the theoretical concepts presented in Chapter 4 in a demo application presentation. The application manages entities from a library such as: books, authors and members. The conceptual model of the application is written in .NET, but the database component of this application is written in Java, then it have to use proxy objects written in Java for the real library model which is written in .NET. The beauty of the solution resides in offering the possibility to save and load .NET objects using the Java version of Hibernate and Java proxy objects for the real .NET objects. It is not necessary to use the NHibernate framework.

Chapter 4 is structured as follows: the first section presents the problem statement together with: (1) the motivation of solving it, (2) current approaches and (3) their drawbacks, the second presents the proposed approach and a proof of concept has been presented in section three. Section four applies two evaluation metrics for the proposed code generation solution. Finally, contributions of this work are presented together with future improvements and open problems.

Chapter 5 presents a technique inspired from hardware development that can be applied in software engineering in order to develop flexible and modular systems based on independent components. On the other hand, it presents the advantages and constraints of this technique and how to implement it in an actual development platform, for instance Java and .NET.

In the hardware development process, an engineer can use many integrated circuits in order to develop a complex system, for instance he/she can use multiplexers, counters, logic AND, logic OR, memory, register [76, 54, 85, 61, 52, 71, 50, 129, 93] etc, these
components are interconnected on a main board and become a complex system. The counter, the multiplexer or other components are developed by a third party company, and they are not dependent by components that produce/consume their inputs/outputs. This principle can be applied both on the hardware and software development process, for example a software system can have many independent components that should be interconnected on a main board in order to become a complex system. These components can be developed by a third party company. Nowadays there are many third party components that are used in software systems, but the connections between them are made by a developer, and when a person should write a program that will use the third party components, this can introduce many errors, more than this, changing the code written by a developer is an expensive process that needs time and money.

This approach presents a technique that allows the automatic code generation for connecting different objects. This technique implies constraints for object classes in order to support the connecting process. This chapter combines the presented theoretical concepts with the notions introduced in Chapter 4 in order to exemplify the benefits of the proposed solutions. The main benefit of the solution proposed in Chapter 5 is: a software component design technique which allows the possibility to write independent components that will be automatically connected using a source code generator. The development process has three main steps: (1) design the system in a graphical environment, (2) generate the skeleton source code and (3) implement the component functionalities.

Chapter 5 is structured as follows: the first section presents the open problem in three subsections: (1) Problem motivation, (2) Current approaches and (3) drawbacks of current approaches; section 2 shows the proposed solution in four subsections: (1) Conceptual View, (2) Glossary, (2) Architecture and (3) How it works; section 3 presents the Building Blocks Dev Studio into four subsections: (1) the main architecture, (2) component communication mechanism, (3) a component-based development approach based on the proposed solution and (4) a case study which contains a demo application developed using the Building Blocks Dev Studio and Indep tools [103, 104]; the fourth section presents two evaluation metrics for the proposed solution and finally section five summarizes the contributions of this work and future improvements.

Chapter 6 presents my research during the three months of mobility to the University of Debrecen from October to December 2010 [107, 105, 106]. Together with professor Adamko Attila, we developed a domain-specific language (DSL) for data intensive application. The proof of concept resides in an Eclipse plug-in which can be used in order to specify applications and to transform them to .NET web applications. The benefits of the proposed DSL resides in the possibility to write transformations to many platform specific models not only to web. Finally a comparison with WebML [109, 86, 123, 98, 79, 17, 21, 133] and WebDSL [29, 56, 48, 49, 47] is presented.
The research presents a DSL for specifying data intensive applications. Using the proposed DSL a PIM of an application can be defined, then using transformations to given PSM the source code can be automatically generated. The DSL has been implemented using Eclipse Xtext, XPand and MWE2, based on documentations from [40, 81, 64, 74].

The terms PIM and PSM [26] are most frequently used in the context of the MDA approach. The key concept is that it should be possible to use a MTL [35, 69, 82, 131] to transform a PIM into a PSM [112, 31, 115, 82, 131].

This chapter proposes two transformation engines: (1) a PIM to .NET web applications engine and (2) a PIM to PHP Code Igniter applications engine. These engines are proofs of concept presented into the Chapter 6, they demonstrate the possibility to transform the platform independent model written using the proposed DSL to many platform specific models.

Chapter 6 is structured as follows: section 1 presents the open problem in two subsections: (1) problem motivation and (2) drawbacks of current DSLs for data intensive applications; section 2 shows the technical details of the proposed solution into four subsections: (1) conceptual view of the solution, (2) technical details, (3) how the solution works and (4) a comparison with WebML and WebDSL. The DevDSL Eclipse plugin, which is a proof of theoretical concepts, is presented in section 3 with the following subsections: (1) the main architecture, (2) how it works; section four presents a list of case studies which contains two demo applications [105] developed using the proposed DSL, a custom transformation engine to PHP Code Igniter framework and a custom validation mechanism for user inputs [106].

Finally, Chapter 7 presents the conclusions and future work. The intent of MDE are: (1) improve software quality; (2) reduce the development time and budget by using artifacts generators engines such as code generators, test cases generators etc; (3) reduce complexity, and (4) improve reuse by enabling developers to work at higher levels of abstraction and to ignore irrelevant details [7]. In this context, current research has been focused on three open problems and proposed conceptual and technical solutions.

Each of the proposed solutions are accompanied by developed tools. The proposed tools are proof of concepts presented as personal contributions. The following list presents a short description of them:

- **Indep**, presented in Chapter 4 is a framework developed in order to use Java objects in .NET and vice-versa. This framework is a proof of platform interoperability concept presented in [108]. Indep framework provides: (1) the source code generator for proxy objects and (2) the communication infrastructure between Java and .NET.

- **Building Blocks Dev Studio**, presented in Chapter 5 is a stand alone application developed in .NET. It is a proof of component-based development concept presented in [102], and can be used together with **Indep** for developing component-based applications with components written in Java and .NET. The software solution has:
(1) a graphical user interface environment for designing the components and links between them and (2) an automatic code generator engine for components skeletons.

- DevDSL is an Eclipse plugin which implements the theoretical concepts presented in [107]. This plugin offers: (1) a specification module which can be used in order to write the application model and (2) a transformation engine which is used in order to generate the .NET source code from the platform independent model written with the specification module.

I would like to express my gratitude to my supervisor, professor Bazil Pârv, for his competent guidance, suggestions and encouragement. I would like to thank to professor Adamko Attila, professor Simona Motogna, professor Militon Frentiu, professor Ioan Lazăr and to all my collaborators for their precious help and guidance. I am also grateful to all the members of Computer Science Department for their helpful suggestions.

I wish to thank for the financial support provided from programs co-financed by the SECTORAL OPERATIONAL PROGRAMME HUMAN RESOURCES DEVELOPMENT, Contract POSDRU 6/1.5/S/3 – “Doctoral studies: through science towards society”.
Bibliography


[76] Stephen Kou and Jens Palsberg. From oo to fpga: fitting round objects into square hardware? In Proceedings of the ACM international conference on Object oriented
programming systems languages and applications, OOPSLA ’10, pages 109–124, New York, NY, USA, 2010. ACM.


[106] Paul Horatțiu Stan. A custom validation mechanism for DevDSL. Submitted to: Studia UBB, Informatica, LVII(1); 2012.


