Experiments in Run-Time Model Extraction

Frédéric Jouault, Jean Bézivin, Régis Chevrel
ATLAS Group (INRIA & LINA, University of Nantes)
{ f.jouault | jbezivin | chevrel.regis }@gmail.com

Jeff Gray,
University of Alabama at Birmingham
gray@cis.uab.edu

Abstract

Reverse engineering and software evolution are probably today among the most challenging research areas in software engineering. They are also very rich fields for applying model engineering techniques. However, in order to define a sound model-driven reverse engineering framework, we need to understand more clearly the conditions under which various abstract models may be extracted from legacy systems and from other various software assets. The operation of extracting a model from a system is still very poorly understood. As part of several projects that have been performed in the INRIA ATLAS team, we have been extracting various kinds of models from various kinds of legacy systems like COBOL, Java, Smalltalk, Visual Basic, etc. Taking stock on this, we study in this paper some of the conditions that would allow partial automation of model-based reverse engineering. More precisely we propose a method for metamodel-driven model extraction. The proof of concept is based on a recent experiment using Visual Basic 9.0 and several previous experiments done with the Squeak version of the Smalltalk language. Besides metamodel-driven model extraction, one of the research contributions of this work is to show the feasibility of extracting models not only from static systems but from dynamic systems as well. In certain ideal conditions, the models extracted from both situations may also be jointly used.

General terms

Reverse engineering, model extraction

Keywords

Parametric metamodel, tagged metamodel, dynamic model, MDRE

1. Introduction

Model Driven Engineering (MDE) promotes the usage of models as first class entities. Any model conforms to a precise metamodel. Some models may be obtained from other models by chains of transformations but the first ones (initial models) have to come from some specific place. In forward engineering, many initial models are created by a human operation. A given situation (for example a business system) is observed by a human agent and the result of this may later be subject to full automation (by the way of model transformations), possibly leading to executable code production. However, mainly in reverse engineering, initial models may sometimes be produced by automatic means. This is for example the case when a model is created from legacy code by a text to model extraction operation.

In order to reach conclusions as advanced as possible, we base our experiments on the ideal context of a legacy written in a language with good reflective capabilities. When there are no introspection capabilities available, the possibilities are obviously much reduced but the general extraction objectives may be partially met by other means.

Besides metamodel-driven model extraction, one of the research contributions of this work is to show the feasibility of extracting models not only from static systems but from dynamic systems as well. In certain ideal conditions, the models extracted from both situations may also be jointly used. To this end it is necessary to have a clear and clean definition of *system*, *model*, and *model extraction*. One contribution of this work is to propose a very general definition of the model extraction operation that could be usable through different contexts and situations.

2. Systems and models

We base our work on the definition proposed in [3]. A terminal model represents a system and conforms to a metamodel. The corresponding relations of *conformance* and *representation* are illustrated in Figure 1. The left part of the figure represents the real

world (or world of discourse). The right part of the figure represents the modeled world. Entities of the real world are *systems* and we represent them by circles. Entities of the modeled world are *models* and we represent them by rectangles.

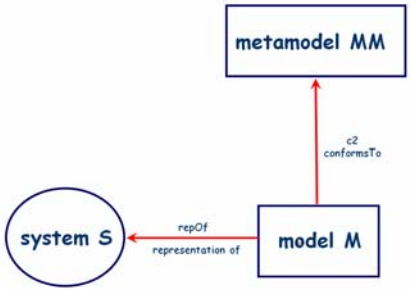

Figure 1 The two basic relations of *representation* and *conformance*

The relation of *conformance* itself may be defined in term of a function μ associating metaelements to elements as illustrated in Figure 2. The relation of *representation* is related to ontology engineering and more difficult to characterize.

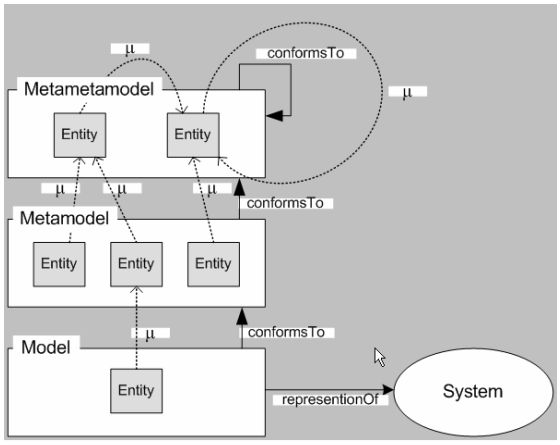

Figure 2 Relations between models and metamodels

A model may be used as a blueprint to produce a system. This is what we may call the operation of *system construction* from a model. We are not interested by this operation in the context of this paper. Conversely a model may also be extracted from the observation of a system. The common property of these two operations (*system construction* and *model extraction*) is that they have a similar post-condition. At the end of both operations the representation relation *repOf (M,S)* holds. As illustrated in Figure 3, the observation of a given system (here a farm) produces a model of this system (here a UML class diagram).

What is not made explicit in Figure 3 is the importance of the metamodel. Here the main features of the modeling languages are *classes* and *attributes* but with another metamodel they could have been as well *event*, *constraints*, *functions*, etc.

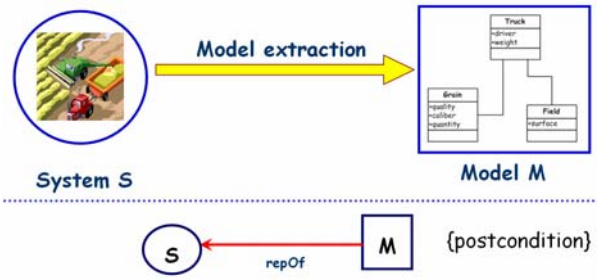

Figure 3 The operation of model extraction and its postcondition

As stated before, many operations of model extraction are made by human operators and are not very well understood. However in some cases it is possible, at least partially, to automate the operation of model extraction. This paper proposes an initial investigation in this area.

There are many categories of models based on their metamodels. But there are also different categories of systems and this is very important for the characterization of the model extraction process. Most systems are dynamic, i.e. they evolve in time. Some systems are static, i.e. they stay always similar.

3. Model extraction from legacy systems

We are going in this section to apply the presented notions to the problem of legacy extraction. The main idea is the following: the code *Ps* of a program (i.e. of a COBOL program) may be considered as a static system. We make no difference here whether the code is printed code or is a file on a hard disk. Its main characteristic is that it is composed of structured static text. Now we may consider as a different system, one particular execution *Px* of this program, in a given context, with specific conditions and input data. Typically this is a dynamic system. It may have a start time and end time of be of indeterminate duration.

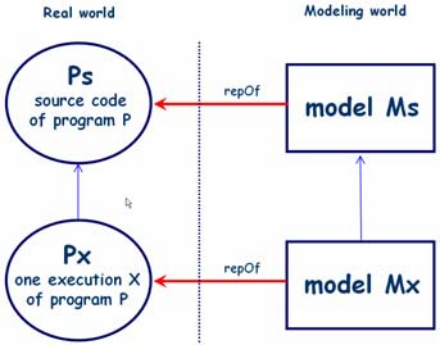

Figure 4 A program and one execution of this program

The classical way to extract a model from legacy is to consider the code as a static system and to apply a model extraction operation to build a model of the code. This has been done in various contexts, with various tools, for several languages like COBOL, Java, C#, etc. A metamodeling system to extract models from COBOL source programs is described in [2] for example. A syntactical analyzer (TGen) was used to build an abstract representation (model) of the source COBOL. Views could be extracted from the resulting model by model transformations in order to locate Y2K or Euro conversion problems. The metamodels used were mainly variants of COBOL metamodels including control structure and data structures. This allowed for example applying slicing algorithms at the model level.

There is another interesting possibility, which is extracting a model from the execution of a legacy program. Let's suppose we have a running Java program, and we would like to extract an execution trace from this particular execution (trace of method calls, of exception risings, of thread activations, etc.). We may consider the (possibly infinite) execution trace of this program as a model. The model extracted from the executing program should conform to a given metamodel. Obviously there are plenty of metamodels that could be used here for example trace metamodels based on different kinds of events.

The illustration of Figure 4 shows that the static source program and one execution of the program may be considered as two different phenomenon of the real world. Obviously we know that they are constrained in the sense that the execution follows the pattern of the source program itself. In the modeled world, there are two models that have also some relation together. In the static legacy model extraction we have only used the upper part of the figure. The lower part can also be used by itself. Many applications will also find convenient to use jointly *Mx* with *Ms*.

4. The case of reflective languages

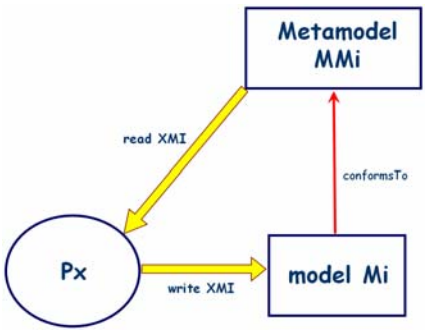

Figure 5 Variable metamodel run-time analysis

Now we are going to consider one more specific kind of system: dynamic execution of a program written in a *reflective language*. If we refer to Figure 4, the new situation here is that *Ps* can be totally inferred from *Px*. We fully use here this capability of *introspection*.

The initial scheme that has been implemented in [5] was to use the Squeak variant of Smalltalk to build the first demonstrator. The principle is illustrated in Figure 5. A Squeak execution *Px* is launched and starts by reading the XMI-serialized form of a metamodel *MMi*. Then, on a given signal (immediately by default), the program writes out the XMI-serialized form of a model *Mi* conforming to the metamodel *MMi*.

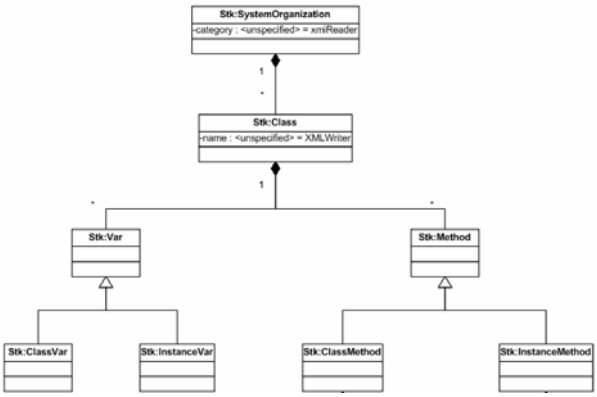

Figure 6 Non decorated Smalltalk metamodel

In this initial experiment, the goal was to use execution *Px* mainly to have access to source program *Ps*. Our original intention was to use introspection instead of syntactical analysis for model extraction from legacy code. Very soon we discovered that the potential was much higher if we could extract run-time models as well.

The effort mainly concentrated on the technical difficulties of XMI reading and writing and finishing the proof of concept prototype. The result of this project was to demonstrate that introspection techniques could be jointly used with metamodeling techniques to allow code to model extraction in a way completely different from classical syntactical analysis techniques.

Of course there was no magic in this solution. If the Squeak program was able to produce a model of itself, according to an arbitrary metamodel, this was because all the information were available. A typical Smalltalk metamodel is presented in Figure 6 with classes, variables, methods, etc. What we wanted was that the Smalltalk program could discover by itself all its classes, all their variables and methods, etc. The solution was to decorate the elements of the metamodel with special comments (like OCL assertions) giving the exact code necessary to discover the corresponding entities. This code was Smalltalk code using the

reflective API of the language. Obviously the decorations were produced beforehand as was the exploration order of the metaentities.
The advantages of this scheme are very numerous since it is quite easy to define a new decorated metamodel.

Since this first experiment many improvements have been made and a new prototype started. The present state of this new prototype has been presented in [6]. The main process is illustrated in Figure 7. It uses another reflective language (Visual Basic Version 9.0) on top of Dotnet. It allows dealing with static and dynamic models of an execution of VB. The metamodels are now expressed in KM3 [1] which is much more readable. The decorations of the metamodels are going to be handled by the technique of model weaving as available in the Eclipse AMW project.

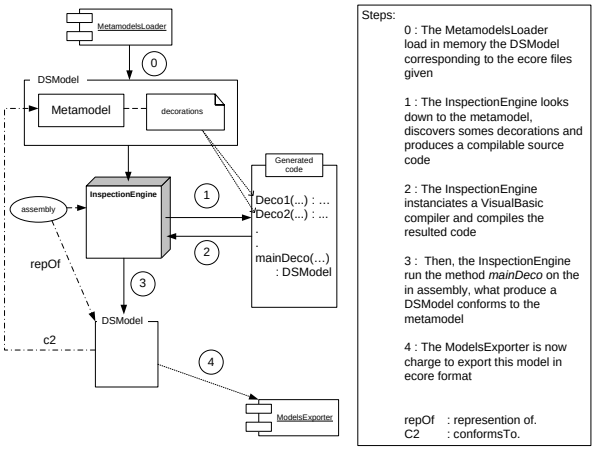

Figure 7 VB Model Extraction mechanism steps

5. Conclusion

In this paper we have seen the power of combining metaprogramming and metamodeling techniques. We have mainly shown how introspection may be combined with advanced model driven engineering to produce powerful software evolution solutions. The possible extension of run-time model to intercession has not yet been studied.

The present work has been done with the goal in mind of contributing to new solutions in the area of model-driven reverse engineering. Some of the solutions will be extended to deal with the objectives of the OMG ADM group (Architecture-Driven Modernization task Force). However some findings in this work goes beyond this scope and show the high potential of run-time models if we are able to provide a regular conceptual framework. It seems to us that more work is needed on the definition of the extraction operation between a system and a model. Our conviction is that this should not be confused with the simple application of model transformation. One possible area would be to investigate the relations of this with the concept of technical spaces [4].

6. Acknowledgements

This work is being partially supported by the ModelPlex project. We thank all the students that have previously worked on related projects, and particularly Julien Blin, Eric Malespine, Guillaume Tillet, Thomas Woerly who achieved the first Squeak implementation in 2003.

7. References

[1] ATL, ATLAS Transformation Language Reference site http://www.sciences.univ-nantes.fr/lina/atl/ including KM3: Kernel Metametamodel definition.

[2] Bézivin, J. sNets: a First generation Model Engineering Platform. In: Lecture Notes in Computer Science, Volume 3844, Satellite Events at the MoDELS 2005 Conference, edited by Jean-Michel Bruel. Springer-Verlag, Montego Bay, Jamaica, pages 169--181.

[3] Bézivin, J., Jouault, F., Kurtev, I., and Valduriez, P., Model-Based DSL Frameworks. OOPSLA'2006 Companion Proceedings, Portland, OR.

[4] Bézivin, J., Kurtev, I., Model-based Technology Integration with the Technical Space Concept. In: Metainformatics Symposium 2005, Esbjerg, Denmark, November 2005, LNCS publication.

[5] Blin, J., Malespine, E., Tillet, G., Woerly, T. TER Report, june 2003, University of Nantes.

[6] Chevrel, R., Bézivin, J., Brunelière, H., Jossic, A., Piers, W., Jouault, F. ModelExtractor: an Automatic Parametric model extractor, ECOOP Workshop on Object-Oriented Reengineering, Nantes, July 2006.