Evolution in Model-Driven Engineering

Louis M. Rose

May 20, 2009

Abstract

To be completed.

This report comprises XX words, as counted by detex | wc -w.

Contents

1 Introduction

Software evolution is a development activity in which a software system is updated in response to some external pressure (such as changing requirements). [Brooks 1986] observes that engineering increasingly complicated systems with traditional development approaches presents many challenges including a resistance to evolution. Software evolution is discussed in Section 1.1.

Model-Driven Architecture (MDA) is a software engineering framework defined by the Object Management Group (OMG) [OMG 2008]. MDA provides a set of standards for developing computer systems in a model-centric, or *model-driven*, fashion. In *model-driven engineering* (MDE), models are utilised as the primary software development artefact. Several approaches to MDE are prevalent today, such as [Stahl *et al.* 2006], [Kelly & Tolvanen 2008] and [Greenfield *et al.* 2004]. The approaches vary in the extent to which they follow the standards set out by MDA.

In MDE, the primary software development artefacts are *models*. A model is a description of a phenomenon of interest [Jackson 1996], and may have either a textual or graphical representation. A model provides an abstraction over an object, which enables engineers of differing disciplines to reason about that object [Kolovos *et al.* 2006]. Employing the MDA guidelines has been shown to significantly improve developer productivity and the correctness of software [Watson 2008].

1.1 Software Evolution in Model-Driven Engineering

In the past, studies [Erlikh 2000, Moad 1990] have found that the evolution of software can account for as much as 90% of a development budget; there is no reason to believe that the situation is different now. [Sjøberg 1993] identifies reasons for software evolution, which include addressing changing requirements, adapting to new technologies, and architectural restructuring.

Software development often involves constructing a system by combining numerous types of artefact (such as source and object code, build scripts, documentation and configuration settings). Artefacts depend on each other. Some examples of these dependencies from traditional development include: compiling object code from source code, generating documentation from source code, and deploying object code using a build script. When one artefact is changed, the development team updates the other artefacts accordingly. Here, this activity is termed *migration*.

MDE introduces additional challenges for controlling and managing evolution [**?**]. For example, MDE prescribes automated transformation from models to code. Transformation may be partial or complete; and may take a model direct to code or use intermediate models [Kleppe *et al.* 2003]. Any code or intermediate models generated by these transformations are interdependent with other development artefacts – e.g. a change to a model may have an impact on other models [Deursen *et al.* 2007]. The process of maintaining consistency between interdependent models is termed *model synchronisation*.

To specify a transformation for a model, the model must have a well-defined set of structural elements and rules that it must obey. This information can be specified in a structured artefact termed a *metamodel*. A model is dependent on its metamodel. When a metamodel is changed, its models may require migration. The process of maintaining consistency between a model and its metamodel is termed *(model and metamodel) co-evolution.*

1.2 Research Aim

In traditional software development, some migration activities can be automated (e.g. background incremental compilation of source code to object code), while some must be performed manually (e.g. updating design documents after adding a new feature). MDA seeks to reduce the amount of manual migration required to develop software, but presently no tools for MDE fully support automated evolution; managing evolution in the context of MDE remains an open research problem.

The aim of our research is to develop structures and processes for managing evolutionary changes in the context of Model-Driven Engineering. In our progress report, we presented and discussed data from existing MDE projects. The analysis of this data led us to conclude that little data was available for studying model synchronisation. Consequently, we decided to focus on developing structures and processes for managing (model and metamodel) co-evolutionary changes.

2 Proposed Thesis Structure

We now discuss the structure of the proposed thesis, highlighting completed and remaining work.

2.1 Introduction

The introduction will be based on the introduction and literature review chapters of my qualifying dissertation. However, Model-Driven Engineering (MDE) is an emerging discipline, and so any content taken from my dissertation will require updating.

2.1.1 Model-Driven Engineering

The proposed thesis will begin by introducing the challenges that MDE addresses. Subsequently, terminology relevant to MDE will be introduced (including terms such as *model*, *metamodel* and *model transformation*). The benefits of MDE will be discussed, along with the main threats to its adoption. These threats include the additional challenges for controlling and managing software evolution with MDE [**?**].

2.1.2 Software Evolution

The introduction will then discuss software evolution, and its causes. The challenges presented by software evolution will be highlighted, particularly in the context of MDE, and used to motivate the proposed thesis.

2.1.3 Research Aim

The high-level aim of the research will be stated, providing a context for the background and literature review chapters.

2.1.4 Research Method

This section will discuss the way in which the research was conducted, including a discussion of the evaluation strategy.

2.2 Background

The background chapter will serve two purposes: Firstly, to introduce areas of computer science that are related to our research, and secondly to introduce two categories of evolution observed in model-driven engineering. These two categories were described in my progress report.

Again, the background section will be based partly on the literature review sections of my qualifying dissertation.

2.2.1 Related Areas

Several subsections will be used, one per related area. Topics are likely to include domain-specific languages and language-oriented programming; refactoring and design patterns; and iterative and incremental approaches to software engineering.

2.2.2 Categories of Evolution in MDE

This section will discuss model and metamodel co-evolution and model synchronisation, two categories of evolution observed in MDE. These categories were introduced in Section 1 of this thesis outline.

2.3 Literature Review

The literature review chapter will provide a thorough review and critical analysis of software evolution research. We will compare and contrast existing techniques for managing and automating activities relating to software evolution. As well as reviewing techniques that apply to the specific challenges caused by software evolution in the context of MDE, we will also critique literature from related areas, such as database and XML schema evolution; and program and modelling language evolution. This chapter will conclude by providing high-level research objectives in the context of the reviewed literature.

2.4 Analysis

The literature review will motivate a deeper analysis of existing techniques for managing evolution in the context of MDE. The benefits and drawbacks of existing techniques will be highlighted by applying them to data from projects using MDE. The analysis will be used to identify requirements for our research.

2.4.1 Locating Data

The first section of the analysis chapter will be based on a section of my progress report, which discusses the data (existing MDE projects) used to analysis existing techniques for managing evolution in the context of MDE. We will introduce and explain the requirements on the data to be used for analysis, identify candidate MDE projects, describe the selection process, and provide reasons for our choices.

2.4.2 Analysing Existing Techniques

Having described the selection of suitable data for the analysis, we will then outline the way in which we have applied existing techniques to the data, and introduce criteria against which the effectiveness of existing techniques will be measured. This work has now been completed, and is discussed in Section **??** of this thesis outline.

2.4.3 Requirements Identification

The analysis of existing techniques will lead to requirements for our research. We will conclude the chapter by enumerating these requirements, refining the high-level research objectives from the literature review chapter into lower-level research objectives.

2.5 Implementation

The implementation chapter will describe the way in which we have approached the requirements presented in the analysis chapter. The requirements will be fulfilled by implementing several related solutions. The solutions will take different forms, including domain-specific languages, automation, and extensions to existing modelling technologies.

2.5.1 Metamodel-Independent Syntax

XMI, an OMG standard for metamodel interchange, and EMF, arguably the most widely used modelling framework, serialises models in a metamodel-specific manner. Consequently, information from the metamodel is required during deserialisation. If the metamodel has evolved since the model was last serialised, deserialisation may fail. This limitation has a major impact on existing techniques for performing co-evolution, forcing them to store old and new copies of the metamodel.

In a submission to ASE 2009, we have highlighted the problems that a metamodel-specific syntax poses for managing and automating co-evolution, and described our solutions. We prescribe the use of a metamodel-independent syntax for storing models. We also show other ways in which a metamodel-independent representation can be useful for managing and automating co-evolution: checking consistency with any metamodel, and performing automatic consistency checking when a new metamodel version is encountered.

2.5.2 Human-Usable Textual Notation

The Human-Usable Textual Notation is an OMG standard textual concrete syntax for the MOF metamodelling architecture. The notation is metamodel-independent – it can be used with any model that conforms to any MOF-based metamodel. HUTN provides a human-usable means for visualising and specifying models, which may not be consistent with their metamodel. HUTN is well-suited to semi-automated migration of inconsistent models.

We have developed the only implementation of OMG HUTN, publishing our work at MoDELS 2008 ([**?**]). We have discussed the way in which HUTN may be used during semi-automated migration of inconsistent models in our submission to ASE 2009.

2.5.3 DSL for Migration Strategies

Some of the requirements presented in the analysis chapter can be addressed with a domain-specific language for specifying migration strategies. Migration strategies will be specified as a model transformation on inconsistent models, which will be expressed in a metamodel-independent representation. Using a metamodel-independent representation affords us some advantages over existing techniques (such as being able to store partially consistent models). A domain-specific language, rather than an existing model-to-model transformation language, is required to address the specific requirements of model migration, as discussed in my progress report.

2.5.4 Further solutions

Further solutions will be required to meet all of the requirements outlined in the analysis chapter. These solutions are likely to include:

- A metamodel refactoring browser for EMF: inspired by the Smalltalk refactoring browser, this tool will provide common refactorings made to improve the design of existing metamodels. Refactorings will preserve model and metamodel consistency.
- Syntax-free modelling:
- Model-driven migration:

2.6 Evaluation

2.6.1 Case Study

2.6.2 Publications

2.6.3 Delivery through Eclipse

2.7 Conclusion

High-level. Summary of research objectives.

2.7.1 Achievement

2.7.2 Future work

3 Progress

3.1 Achievements

3.2 Plan

References

[Brooks 1986] Frederick P. Brooks. No silver bullet – essence and accident in software engineering (invited paper). In *Proc. International Federation for Information Processing*, pages 1069–1076, 1986.

[Deursen *et al.* 2007] Arie van Deursen, Eelco Visser, and Jos Warmer. Model-driven software evolution: A research agenda. In *Proc. Workshop on Model-Driven Software Evolution*, pages 41–49, 2007.

[Erlikh 2000] Len Erlikh. Leveraging legacy system dollars for e-business. *IT Professional*, 2(3):17–23, 2000.

[Greenfield *et al.* 2004] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.* Wiley, 2004.

[Jackson 1996] Michael Jackson. *Software Requirements and Specifications.* Addison-Wesley and ACM Press, 1996.

[Kelly & Tolvanen 2008] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modelling.* Wiley, 2008.

[Kleppe *et al.* 2003] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise.* Addison-Wesley, 2003.

[Kolovos *et al.* 2006] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack. The Epsilon Object Language (EOL). In *Proc. European Conference on Model Driven Architecture – Foundations and Application*, volume 4066 of *LNCS*, pages 128–142. Springer, 2006.

[Moad 1990] J Moad. Maintaining the competitive edge. *Datamation*, 36(4):61–66, 1990.

[OMG 2008] OMG. Object Management Group home page [online]. [Accessed 30 June 2008] Available at: http://www.omg.org, 2008.

[Sjøberg 1993] Dag I.K. Sjøberg. Quantifying schema evolution. *Information & Software Technology*, 35(1):35–44, 1993.

[Stahl *et al.* 2006] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management.* Wiley, 2006.

[Watson 2008] Andrew Watson. A brief history of MDA. *Upgrade*, 9(2), 2008.