

# Model-Driven Evolution of Software Architectures

Bas Graaf

*Delft University of Technology*

*The Netherlands*

*b.s.graaf@tudelft.nl*

## Abstract

*This paper gives an overview of a Ph.D.-project that investigates approaches to support the evolution of software architectures. Particularly, we focus on the use of model-driven techniques in the context of software product-lines. We recognise four tasks related to software evolution: evaluation, conformance checking, migration, and documentation. We propose model-driven solutions for these tasks. By their application in industrial case studies we investigate the extent to which the evolution tasks can be automated, the impact of the use of software product lines, and the possibilities and difficulties to integrate these solutions in industrial practice.*

## 1. Introduction

When changes are required to a software system, the question is whether they can be implemented within the bounds set by the current architecture or require a redesign of the architecture. The former causes types of software evolution referred to as architectural drift and erosion [1]. Eventually, these make a redesign of the architecture unavoidable. In our research we consider the latter type of software evolution, that is, on the level of architecture.

Many companies extended the scope of their software architectures from single systems to multiple systems to increase reuse and reduce required development and maintenance effort<sup>1</sup>. These companies develop a whole range of products that have much in common. By developing such products as a software product line [2] their commonalities and variabilities are made explicit in a product-line architecture. The development of individual products is reduced to specifying the required variation, that is, if *all* variability is made explicit in the product-line architecture. A first step to arrive at such a mature product-line, is the identification of all commonalities and their implementation as a (domain-specific) software platform [3]. All product-line members are built on top of that platform.

To hide the behavioural and structural complexity of such platforms, model-driven engineering (MDE) ap-

proaches are introduced [4]. MDE refers to software development approaches in which models are considered the primary development artefacts [5]. In these approaches software models are gradually transformed (automatically) into source code that runs on top of a platform.

The foundations for MDE are abstraction (modelling) and automation (model transformations). In software engineering the purpose of a model is to represent a particular aspect of a software system. A model can be used either descriptively to determine properties of a system, or prescriptively as a specification of a system to be built [6]. The trade-off involved with modelling is that a model should be simpler (to use and create) than the real system, but complete enough to be useful.

In our work we address the problem of software evolution in the context of MDE and product-lines.

## 2. Problem

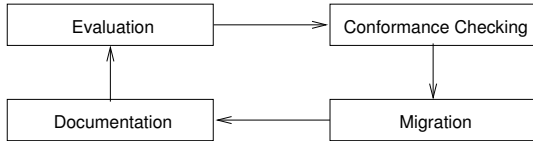
Currently, MDE approaches, such as OMG's Model Driven Architecture (MDA [7]), are focussed on the generation of code from models. The type of model transformations involved are typically vertical (from abstract to concrete). Most software development activities, however, concern software maintenance and evolution [8]. In contrast, these activities involve, for instance, also horizontal transformations, such as the migration of systems from one platform to another.

Therefore, we focus on the evolution of software platforms and the systems they support. More particularly, we address the problem of their evolution on the architectural level. A final focus point is that we, in-line with MDE, aim at the development of automated techniques, where possible.

The solution of software engineering problems in practice has been hampered by industry's resistance to adoption of state-of-the-art software engineering technologies. An important reason for this is that such technologies often have a large impact on current ways of working, resulting in unacceptable risks [9]. This means that, in the context of software evolution, we have to take into account, for instance, the informal use of modelling languages in industry [10]. This makes automation particularly difficult. In general, the impact of solutions

---

<sup>1</sup>For examples, visit the Product Line Hall of Fame: [http://www.sei.cmu.edu/productlines/plp\\_hof.html](http://www.sei.cmu.edu/productlines/plp_hof.html)



**Figure 1. Software evolution tasks**

(technologies or processes) to current ways of working should be minimised.

To limit the scope of our work we restrict ourselves to four types of activities related to evolution of software. We refer to these activities as software evolution tasks. The tasks we consider are depicted in Figure 1 in an evolutionary software life-cycle and are explained below.

**Evaluation** In particular we consider the assessment of whether a product-line architecture requires changes in the face of anticipated changes to the product-line members. Dobrica and Niemelä [11] gives an overview of proposed architecture evaluation approaches. However, none of those is explicitly aimed at software product lines.

**Conformance Checking** In the case that an evaluation indicates that architectural changes are required, knowledge of the extent to which ‘downstream’ development artefacts (e.g., product-line members) conform to the software (product-line) architecture is valuable. Krikhaar [12] and Mens [13] compare a number of approaches to check architecture conformance. However, conformance between models at different abstraction levels is not addressed. Moreover, most approaches dictate the introduction of specific modelling languages, requiring a change to current ways of working.

**Migration** The migration to a new product-line architecture and associated software platform that better supports foreseen requirements, requires the migration of all products supported by the legacy platform. There is no previous work that considers software (architecture) migration as a model transformation problem. Several other works do address the transformation of software systems. However, they consider single-product architectures [14], simple graphs [15], or the level of source code [16]. The language migration process used by Terekhov and Verhoeve [16] is particularly interesting. It separates a migration in three phases that include restructuring of source programs to enable the (automatic) transformation phase. Although it was used for source code migration, such a preparatory step is also required for the migration on the architectural level to take into account idiosyncratic industrial modelling conventions.

**Documentation** After a migration of the (product-line) architecture and the products-line members it supports, documentation needs to be updated. It is generally accepted that the documentation of software architectures consists of multiple views [17]. Often UML is used in these views. On the other hand, architecture description languages (ADL’s) and MDE support the creation of models to automate several software engineering tasks, such as code generation. However, no approach addresses the problem of keeping documentation and models consistent. With the upcoming of MDA and other MDE approaches this becomes a highly relevant problem.

### 3. Research Questions

We investigate the integration of model-driven and architecture-driven software development approaches and their deployment for software evolution tasks. As such, our main research question is:

*How can model-driven software engineering approaches be deployed to support software architecture evolution?*

When considering the problem description in Section 2, this question raises the following sub-questions:

- To what extent can this support be automated?
- What is the impact of the use of software product lines and platforms on this support?
- How to integrate this support in practice, considering the informal use of modelling languages and preference for proven technologies in industry?

The relevance of our work lies in the fact that we take into account several advances in software development practices, that is, software product lines and MDE. At the same time we consider these approaches in terms of software evolution tasks. It is those tasks that take up most of the time, effort, and money of software development projects and organisations [8]. Model-driven support at the architectural level for these tasks allows for (partial) automation, resulting in improved reliability, efficiency, and quality.

### 4. Research Methods

We intend our research to be industry-driven. Therefore, we adopt the ‘industry-as-laboratory’ approach proposed by Potts [18]. We accomplish the interactions with industry on which this approach is based in three ways: a survey, industrial case studies, and close collaboration with software practitioners in industry.

We first perform a survey among more than 35 software practitioners at eight companies to get an overview

of software engineering practices and specific problems in the (embedded) software industry [9]. The observations made in that survey, include the upcoming use of product-line approaches, the informal use of modelling languages, and the importance of the evolutionary aspect of software. This survey partially determines the problems we address in the research described in this paper.

Our type of research questions, makes it is difficult to identify and measure controlled variables. Furthermore, as our research is industry-driven, we want to avoid scalability problems. Therefore, as suggested by Kitchenham et al. [19], we use case studies to investigate the applicability of model-driven approaches to the software evolution tasks we defined. For each task we propose a separate solution, which we evaluate in a (industrial) case study. The case studies we conducted are mainly related to two industrial systems.

We qualitatively evaluate the solutions we propose by carefully observing and analysing their application in the case study. For the migration case study we were able to compare our findings with respect to a migration of the same system conducted manually. The conformance checking tasks were only executed using our techniques. Therefore, their evaluation is based on the type and number of inconsistencies found. For the evaluation and documentation tasks, we evaluate our solutions with respect to the application of similar approaches in other cases.

Considering our research questions, we specifically focussed the evaluation on the extent to which the software evolution tasks can be automated, the impact of software product lines, and possibilities for reusing (proven) software technologies and reducing organisational impact.

## 5. Solution

For the first of our software evolution tasks, we defined a process-based solution based on the Software Architecture Assessment Method (SAAM [20]). Our approach takes into account the product-line aspect by the introduction of different types of scenarios. Furthermore, by reducing the amount of effort required from different stakeholders, the organisational impact of the approach is minimised.

In contrast to our solution for the evaluation task that does not necessarily requires models, our solution to the remaining tasks are model-driven. As architecture evolution involves the change of architectural models, we propose to consider those software evolution tasks as model transformation problems. Model-driven support for these tasks involves the definition of metamodels and model transformations. In our work we use MDA technologies, such as the Unified Modeling Language

(UML, for modelling), the Meta-Object Facility (MOF, for metamodelling), XML Metadata Interchange (XMI, for model interchange), and the Atlas Transformation Language (ATL, for model transformations).

We focus on two issues related to conformance checking. One is the semantic gap between models at different levels of abstraction and between models and code. We address this by the definition of conformance models at an ‘intermediate’ abstraction level. The other issue is the integration of conformance checking in practical (model-based) development processes. This not only requires to use industry standards, but also to take into account (informal) industrial modelling conventions.

We address these issues by demonstrating how to implement conformance checking using XML and MDA technology and the introduction of a normalisation step. Together, with the generation of conformance models this allows our approach to be applied with minimal impact on existing development processes.

Our migration approach also involves the definition of suited metamodels for source and target of the migration. It defines a migration as separate normalisation and transformation steps. Again, the normalisation step is required because of the informal use of modelling languages in industry. The transformation step is specified and implemented using a model transformation language. Finally, the involved normalisation and transformation rules are defined for a set of key concerns for the particular domain.

For documentation of software architectures we rely on the use of UML as described by Medvidovic et al. [21]. In our approach, however, we allow at the same time the use of ADL’s for other software engineering tasks. By using model transformations to create an explicit mapping between ADL models and documentation using UML, the approach enables their simultaneous evolution.

## 6. Results and Contributions

Our results demonstrate the applicability of model-driven solutions to specific software evolution tasks. For all software evolution tasks we considered, we proposed solutions that take into account product-line architectures (opposed to single-product architectures), aim to reduce organisational impact, or are model-driven. Furthermore, we extend and use technologies that have already proven their applicability in practice, such as SAAM and MOF.

Although our solutions were investigated in the context of concrete (industrial) problems, our evaluations show that they can be applied to these software evolution tasks for a broader class of systems. Moreover, as they are accompanied by stepwise processes, we are able to indicate which part is automated. In particular, the main

contributions of our research are:

- An overview of the software engineering technologies used in embedded software industry [9]
- An approach for the evaluation of product-line architectures [22].
- An XML- and view-based approach for checking the conformance of implementation and architecture [23].
- A model-driven approach for checking the conformance of state-based and interaction-based behavioural models [24].
- A model-driven approach for the migration of supervisory machine control architectures [25]
- A model-driven approach for simultaneous evolution of models and documentation based on views, UML, and MDA.

At the time of writing, we are in the final stage of our work. All case studies have been performed and most of the results have been published in international workshops, conferences, and journals [9, 22, 23, 24, 25].

**Acknowledgement** We received partial support for our work via the Reconstructor (NWO), and MOOSE (Sender) projects.

## References

- [1] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.
- [2] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.
- [3] Jan Bosch. Maturity and evolution in software product lines: Approaches, artefacts and organization. In *Proc. 2<sup>nd</sup> Int'l Conf. Software Product Lines (SPLC 2)*, volume 2379 of *Lecture Notes in Computer Science*, pages 257–271. Springer-Verlag, August 2002.
- [4] Douglas C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2):25–31, February 2006.
- [5] Jean Bézivin. On the unification power of models. *Software and Systems Modelling*, 4(2):171–188, May 2005.
- [6] Ed Seidewitz. What models mean. *IEEE Software*, 20(5):26–32, September 2003.
- [7] OMG. MDA. <http://www.omg.org/mda>, 2007.
- [8] B. P. Lientz, E. B. Swanson, and G. E. Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6):466–471, 1978.
- [9] Bas Graaf, Marco Lormans, and Hans Toeteneel. Embedded software engineering: The state of the practice. *IEEE Software*, 20(6):61–69, November–December 2003.
- [10] Christian F.J. Lange, Michel R.V. Chaudron, and Johan Muskens. In practice: UML software architecture and design description. *IEEE Software*, 23(2):40–46, March 2006.
- [11] L. Dobrica and E. Niemelä. A survey on software architecture analysis methods. *IEEE Trans. Software Engineering*, 28(7):638–653, July 2002.
- [12] René L. Krikhaar. *Software architecture Reconstruction*. PhD thesis, Univ. van Amsterdam, 1999.
- [13] Kim Mens. *Automating Architectural Conformance Checking by means of Logic Meta Programming*. PhD thesis, Vrije Univ. Brussel, 2002.
- [14] Jan Bosch and Peter Molin. Software architecture design: evaluation and transformation. In *Proc. IEEE Conference and Workshop on Engineering of Computer-Based Systems (ECBS'99)*, pages 4–10. IEEE CS, 1999.
- [15] Hoda Fahmy and Richard C. Holt. Using graph rewriting to specify software architectural transformations. In *Proc. 15<sup>th</sup> IEEE Int'l Conf. Automated Software Engineering*, pages 187–196. IEEE CS, 2000.
- [16] Andrey A. Terekhov and Chris Verhoef. The realities of language conversions. *IEEE Software*, 17(6):111–124, November 2000.
- [17] Christine Hofmeister, Philippe Kruchten, Robert L. Nord, Henk Obbink, Alexander Ran, and Pierre America. Generalizing a model of software architecture design from five industrial approaches. In *Proc. 5<sup>th</sup> Working IEEE/IFIP Conf. Software Architecture (WICSA 5)*, pages 77–88. IEEE CS, 2005.
- [18] Colin Potts. Software engineering research revisited. *IEEE Software*, 10(5):19–28, September 1993.
- [19] Barbara Kitchenham, Lesley Pickard, and Shari Lawrence Pfleeger. Case studies for method and tool evaluation. *IEEE Software*, 12(4):52–62, July 1995.
- [20] Rick Kazman, Gregory Abowd, Len Bass, and Paul Clements. Scenario-based analysis of software architecture. *IEEE Software*, 13(6):47–55, November 1996.
- [21] Nenad Medvidovic, David S. Rosenblum, David F. Redmiles, and Jason E. Robbins. Modeling Software Architectures in the Unified Modeling Language. *ACM Trans. Softw. Eng. Methodol.*, 11(1):2–57, 2002.
- [22] Bas Graaf, Hylke van Dijk, and Arie van Deursen. Evaluating an embedded software reference architecture – industrial experience report. In *Proc. 9<sup>th</sup> European Conf. Software Maintenance and Reengineering (CSMR 2005)*. IEEE CS, 2005.
- [23] Hylke W. van Dijk, Bas Graaf, and Rob Boerman. On the systematic conformance check of software artefacts. In *Proc. 2<sup>nd</sup> European Workshop on Software Architecture (EWSA 2005)*, volume 3047 of *Lecture Notes on Computer Science*, pages 203–221. Springer-Verlag, June 2005.
- [24] Bas Graaf. Model-driven consistency checking of behavioural specifications. In *Proc. 4<sup>th</sup> Int'l Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007)*, 2007. Accepted for publication.
- [25] Bas Graaf, Sven Weber, and Arie van Deursen. Migrating supervisory control architectures using model transformations. In *Proc. 10th European Conf. Software Maintenance and Reengineering (CSMR 2006)*, pages 151–160. IEEE Computer Society, 2006.