

# Software Language Evolution

Sander Vermolen

Delft University of Technology  
The Netherlands  
S.D.Vermolen@tudelft.nl

August 4, 2008

## Abstract

*By abstraction and factoring out domain specific knowledge, model driven engineering addresses the problem of increasing software complexity. Both models and meta models are generally subject to evolution, yet evolution of a meta model can cause conforming models to no longer conform and thereby no longer be usable. Therefore, models need to be migrated to reflect changes to their meta models. As evolution is typically frequent and reoccurring, manual migration of models is cumbersome and holds back the development process, yet automatic support is generally lacking. In this research we identify the problems caused by meta model evolution and develop methodologies and tools to solve these by supporting meta model evolution generically and automatically.*

## 1. Introduction

Model driven engineering (MDE) addresses the problem of increasing software complexity. By abstraction and factoring out domain specific knowledge, it aims at improving developers productivity and understanding of software. MDE proposes models from which software can be generated. Models conform to meta models, which are, in contrast to traditional development, considered to be a development artifact.

Due to changing context or changing requirements, models need to evolve. Similarly, meta models are subject to evolution. However, artifacts that originally conformed to a meta model, may no longer conform after evolution of the meta model and thereby become useless. Artifacts include conforming models, but also transformations on conforming models and generation of software from models.

To enable meta model evolution, conforming artifacts need to be migrated to reflect changes to their meta model. Coupled evolution combines meta model evolution and reflecting migration. Coupled evolution for meta models is shown graphically in Figure 1. The figure consists of two layers. The top layer represents two versions of a meta model and the evolution between them. The bottom layer represents the model that is migrated accordingly. The solid vertical

lines represent conformance, indicating that the meta models describe the structure of the models.

To allow meta models to continuously evolve over time, manual migration of artifacts is insufficient. To support automatic migration, we need a formalism to specify evolution. Several approaches have been proposed [13, 9, 18], most of which are based on definition by identifying small evolution steps. Such steps can be mapped to transformations of underlying artifacts, as represented by the dashed arrow in Figure 1. In practice, evolution is established by editing an existing meta model to create a new one. Having to explicitly specify the edits would be cumbersome and thereby hold back the evolution process. Automatic detection of the evolution would ease the evolution process.

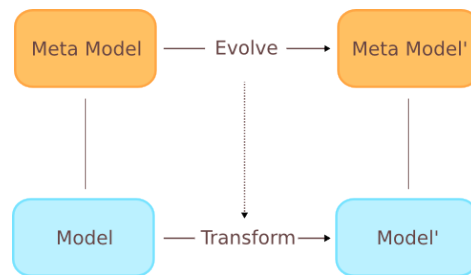


Figure 1. Coupled meta model evolution

Coupled evolution is not restricted to meta models and models. It reoccurs in many domains that hold conforming artifacts. Examples are data models and conforming data, domain specific languages (DSLs) and conforming applications, but also generic programming languages and conforming applications. Furthermore, coupled evolution is applicable to different pairs of levels. It is applicable to meta models and models, but also to meta-meta models and meta models [14]. To generalize, we will use the term software language evolution, which comprises evolving software languages and migration of software.

Despite its frequent occurrence, software language evolution has limited support. Although there is significant work on migrating databases [10, 2, 3], the coupling to data model evolution and automated support is lacking. Furthermore, the

focus is merely on databases, not on meta model or DSL evolution. Consequently, development is held back by the amount of manual effort required to do migration and limited to large, infrequent evolution steps that pose a big risk to the development process.

## 2. Problem Definition

The main focus of this work is on supporting software language evolution. The support involves structured evolution of the software language itself as well as coupled evolution of the various artifacts that conform to the language. We will primarily focus on artifacts that directly conform to the language, namely the software. Artifacts that indirectly conform to the language (e.g. transformations on the software or documentation of the software) are less relevant in our research. The main goal is to support continuous software language development that requires little or no human effort. Automation is the key to achieve this.

In our research, we will mainly try to answer the following questions: What classes of software language evolution can be identified in real-world cases? Is it possible to develop generic methodologies to support software language evolution? To what extent can these technologies be automated to reduce manual effort?

## 3. Approach

In our research we try to develop software language support by (1) exploring when software language evolution occurs and how it is applied; (2) developing methodologies to support software evolution generically; and (3) implementing these methodologies in tools to validate and improve them. In this section we first discuss the exploration of software language evolution. Then we outline the sub domains of our research. Finally we look at what application domains we take into account in validation and tool support.

### 3.1. Exploring Software Language Evolution

Part of our research will be on exploring software language evolution in practice. Through real-world case studies we plan to improve our understanding of realistic software language evolutions. Results from the exploration will be used as basis for development and improvement of methodologies and tools. Specifically, we try to answer the following questions: What types of software language evolution are common and when do they occur? What problems arise in practice from software language evolution? In what ways is software language evolution currently supported in practice?

For exploration studies, we use both industrial and non-industrial cases in the areas of data modeling, meta modeling and DSLs. These case studies will both be used for explorative purposes as well as for validation.

### 3.2. Sub-problems

We identify the following sub-problems in our research:

**Defining software language evolution** Software languages evolve when their definition is edited by the developer. The developer knows and understands the changes and uses his knowledge to migrate models. When automating the process, evolution needs to be specified explicitly.

Several approaches to defining software language evolution exist. We distinguish direct comparison, in which we specify differences between two versions and indirect comparison in which we specify a trace of change operations to the software language [5]. The first is relatively straight forward to specify, but captures little knowledge about the evolution process. The second is much harder to specify, but identifies smaller (sub) steps within a single evolution step. It captures more knowledge on how the evolution occurred and on how it should be handled.

In addition to the structure of evolution specifications, we need to identify what evolution operations exist and how these can be combined to form a bigger evolution step. Reusing and abstracting from evolutions are subjects that need to be dealt with next. Furthermore, considering evolution as a sequence of evolution steps implies the need for ‘coupled versioning’. Artifacts conforming to some (arbitrary) software language version may need to be migrated to some (arbitrary) other version.

**Automatic migration of conforming artifacts** Once evolution of a software language has been defined, conforming artifacts need to be migrated accordingly. To establish automatic migration, we need to construct mappings from evolution specifications to software migrations. As the mappings are specific to the domain of application, we may need to implement several.

**Detecting software language evolution** Depending on the chosen formalism for specifying software language evolution, specification of evolution could become a significant manual effort. Consequently, software language evolution in general would again require too much manual work. We will therefore also focus on automatic derivation of evolution specifications from given software language versions (e.g. two data model versions).

Detecting software language evolution is not merely a difference calculation. Evolution steps are partially captured in language differences but also partially in the ideas of the developer. Consider a data model consisting of entities with properties. Suppose we detect the addition of a property in one entity and the removal of a property in another. Such a combination can indicate the removal of one property and the addition of another, but also moving a single property from one entity to the other.

We roughly distinguish two approaches to evolution detection: Record changes in a software language editor, or de-

tect changes afterwards. Since we do not want to restrict the developer to a certain editor, we will focus on the second.

**Evolution of software language semantics** In general, software language definitions only capture syntax of software. Semantics is not explicitly specified. For example, a data modeling language is usually defined to contain entities and properties. Yet, the meaning of an entity or a property is not explicitly defined. Merely changes to the syntax of a meta model are not always sufficient to derive model migrations from. The addition of a property to an entity specifies what happens to the data model, but when migrating the data accordingly, we also need to know what the value of the new property should be. Data values are only relevant at the data level. It captures some semantics of properties.

Although evolution support for meta modeling currently mainly focuses on evolution of syntax, evolution of semantics is also part of software language evolution and will therefore also be part of our research. Syntax and semantics of software languages are aspects that are hard to deal with separately in the software language evolution context. Along with the introduction of a property, we want to define the computation of its value. Separating the two would require additional effort but may provide improved insight.

### 3.3. Application Domains

Although we primarily focus on supporting software language evolution in general, not all components can be constructed generically (e.g. the mapping from language evolution to software evolution). When implementing these components, we mainly focus on three domains:

**Data modeling** We will be focusing on data models within the WebDSL project [17]. WebDSL is a domain specific language for defining data-driven web applications. It includes a generation from a WebDSL application to an actual web application. Part of the WebDSL language is a strong data modeling sub-language that supports (amongst others) inheritance, object ownership and unidirectional relations. One of the goals of our research is to provide full-automatic database migration to reflect evolution of a WebDSL data model.

**Meta modeling** In contrast to data transformations and data modeling, model transformations are an intrinsic part of meta modeling, which eases the transformation of models, yet may have implications on evolution of the meta model. Meta modeling is generally set up graphically and is thereby likely to require a different type of tool support compared to data modeling. Due to its open development and common acceptance, the Eclipse Modeling Framework (EMF) [4] is suitable to support meta model evolution in.

**DSLs** Closely related to the domain of meta modeling are DSLs. DSLs are generally textual and defined by a grammar formalism, yet the goal is similar to that of meta models.

For DSLs, we will use SDF [15] for grammar definition and Stratego/XT [16] for migration of DSL applications.

## 4. Validation Strategy

Validity of our approach will be assessed by empirical research. We will use case studies [8] to validate our methodologies, discover their limitations and improve our tools and techniques. The case studies will be on real-world applications. By validation we will seek answers to the questions: Are the techniques correct and scalable? What is the feasibility of using the techniques in practice (e.g. what is the degree of automation)? How applicable is the approach to the application domains?

## 5. Status

Currently we are through 20% of the project time line. We have focused on defining software language evolution generically and supporting automatic migration for data modeling in WebDSL. Although WebDSL focuses purely on web applications, the support for data model evolution is more widely applicable.

We have proposed [14] a generalization of coupled evolution to implement generic support for software language evolution. Furthermore, we have proposed an architecture to support software language evolution. It includes: (1) Generation of a domain specific transformation language (DSTL) to specify evolution for a given domain. (2) Generation of an interpreter of transformations in a DSTL. (3) Generic abstractions from the basic transformations that are defined in the DSTL. We have implemented a tool to support the architecture and applied it to the domain of data modeling in WebDSL. The result of the application to data modeling is a tool to automatically migrate databases along with an evolving (WebDSL) data model.

Our work until now mainly covers the sub domains ‘Defining software language evolution’ and ‘Automatic migration of conforming artifacts’. We plan to improve current results by allowing for easier abstractions and improving the integration of data model semantics. Subsequently, we will extend our work by evolution detection and by applying it to meta modeling and DSLs. Furthermore, we will use case studies to improve our insight into software evolution in practice and to validate our methodologies and tools.

## 6. Related Work

Coupled evolution plays a significant role in computer science and has been treated in various areas. Earlier research has primarily focused on constructing coupled evolution support for specific domains. We discuss related work in the

most relevant domains: model evolution, DSL evolution and schema evolution.

Coupled evolution for the meta modeling domain is introduced by Gruschko [5]. As is the case for most publications on coupled evolution for meta models, he models evolution using small elementary transformation steps. Wachsmuth [18] applies coupled evolution to MOF [11] compliant meta models. Ecore, the meta-meta model in EMF, is a variant on MOF. Wachsmuth introduces a set of transformations and proposes a mapping to model migrations implemented in QVT [12]. Similar to Wachsmuth, Herrmannsdoerfer [7] looks at coupled model evolution based on small evolution steps. He focuses on EMF itself. Herrmannsdoerfer provides a prototypical editor based on Eclipse.

In the area of domain specific languages, Pizka et al. discuss the evolution of DSLs. They claim three obstacles in DSL development: (1) the need for stepwise bottom-up generalization of DSLs; (2) DSLs should be layered; and (3) automated co-evolution is required for DSLs. In our work, we try to solve the first and last obstacle. The second obstacle is not directly related to evolution.

With respect to coupled data model evolution (or schema evolution) and the related two-level data transformations, numerous approaches have been found to solve these problems [10, 3, 2, 6, 1]. These mainly focus on the schema to data mapping for a specific type of schema.

## 7. Contributions

Our expected contributions include the development of concepts, techniques and tools to support software language evolution, as well as their validation against real-world scenarios. This includes: A generic methodology to define software language evolution, allowing abstractions and versioning; Automatic migration of software in the domains of data modeling, meta modeling and DSLs; Generic (semi-) automatic detection of evolution; Integration of software language semantics in the coupled evolution process. Furthermore, we expect to contribute the identification and classification of common software language evolution problems in real-world (industrial) cases.

**Acknowledgments** This research is supported by NWO/JACQUARD project 638.001.610, *MoDSE: Model-Driven Software Evolution*.

## References

- [1] T.L. Alves, P.F. Silva, and J. Visser. Constraint-aware schema transformation. In *Ninth International Workshop on Rule-Based Programming (Rule 2008)*, 2008.
- [2] P. Berdager, A. Cunha, H. Pacheco, and J. Visser. Coupled schema transformation and data conversion for XML and SQL. In *Practical Aspects of Declarative Languages (PADL 2007)*, volume 4354 of *LNCS*, pages 290–304. Springer, 2007.
- [3] A. Cunha, J.N. Oliveira, and J. Visser. Type-safe two-level data transformation. In *Formal Methods Europe (FME 2006)*, volume 4085 of *LNCS*, pages 284–299. Springer, 2006.
- [4] Eclipse Foundation. Eclipse Modeling Framework Project (EMF). <http://eclipse.org/emf>, 2008.
- [5] B. Gruschko, D. S. Kolovos, and R. F. Paige. Towards synchronizing models with evolving metamodels. In *Workshop on Model-Driven Software Evolution at CSMR 2007*, 2007.
- [6] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *International conference on management of data (SIGMOD 1993)*, pages 157–166, New York, NY, USA, 1993. ACM.
- [7] Markus Herrmannsdörfer. Metamodels and models. Master’s thesis, München University of technology, München, Germany, July 2007.
- [8] B. Kitchenham, L. Pickard, and SL Pfleeger. Case studies for method and tool evaluation. *Software, IEEE*, 12(4):52–62, 1995.
- [9] R. Lämmel. Grammar adaptation. In *Formal Methods Europe (FME 2001)*, volume 2021 of *LNCS*, pages 550–570. Springer, 2001.
- [10] Ralf Lämmel and Wolfgang Lohmann. Format Evolution. In *Reverse Engineering for Information Systems (RETIS 2001)*, volume 155 of *books@ocg.at*, pages 113–134. OCG, 2001.
- [11] Object Management Group (OMG). Meta Object Facility (MOF) Core Specification - Version 2.0, January 2006.
- [12] Object Management Group (OMG). MOF QVT Final Adopted Specification, March 2007.
- [13] M. Pizka and E. Jurgens. Tool-supported multi-level language evolution. In *Software and Services Variability Management Workshop*, volume 3 of *Helsinki University of Technology Software Business and Engineering Institute Research Reports*, pages 48–67, Helsinki, Finland, April 2007.
- [14] S.D. Vermolen and E. Visser. Heterogeneous Coupled Evolution of Software Languages. Technical Report TUD-SERG-2008-028, Delft University of Technology, Software Engineering Research Group, 2008.
- [15] E. Visser. *Syntax Definition for Language Prototyping*. PhD thesis, University of Amsterdam, September 1997.
- [16] E. Visser. Program transformation with Stratego/XT: Rules, strategies, tools, and systems in StrategoXT-0.9. In *Domain-Specific Program Generation*, volume 3016 of *LNCS*, pages 216–238. Springer, June 2004.
- [17] E. Visser. WebDSL: A case study in domain-specific language engineering. In *Generative and Transformational Techniques in Software Engineering (GTTSE 2007)*, LNCS. Springer, 2008. Tutorial for International Summer School GTTSE 2007; to appear.
- [18] Guido Wachsmuth. Metamodel adaptation and model co-adaptation. In *21st European Conference on Object-Oriented Programming (ECOOP 2007)*, volume 4609 of *LNCS*, pages 600–624. Springer, 2007.