Evolution of Modelling Languages

Bart Meyers, Hans Vangheluwe

Modelling, Simulation and Design Lab (MSDL), University of Antwerp, Middelheimlaan 1, B-2020 Antwerp, Belgium

Abstract

In model-driven engineering, evolution is inevitable over the course of the complete life cycle of complex software-intensive systems and more importantly of entire product families. Not only instance models, but also entire modelling languages are subject to change. This is in particular true for domain-specific languages. Up to this day, modelling languages are evolved manually, with tedious and error-prone migration of for example instance models as a result. This position paper discusses the different evolution scenarios for various kinds of modelling artifacts, such as instance models, meta-models and transformation models. Subsequently, evolution is de-composed into four primitive scenarios such that all possible evolutions can be covered. We suggest that our structured approach will enable the design of (semi-)automatic model evolution solutions[1].

1. Introduction

In software engineering, the evolution of software artifacts is ubiquitous. These artifacts can be programs, data, requirements, documentation, but also languages. Language evolution applies in particular to domain-specific modelling (DSM), where domain-specific languages (DSLs) are specifically designed to minimize accidental complexity by using constructs closely coupled with their domain. This results in a reported productivity increase of a factor 5 to 10 [9]. DSLs must be quickly built and used, and grow incrementally. A formal underpinning for DSM is given by multi-paradigm modelling (MPM) [12].

The high dependence on their domains and the need for instant deployment make DSLs highly susceptible to change. Such an evolution of a language can have substantial consequences, which will be explained throughout this paper. Early adopters of the model-driven engineering paradigm dealt with this evolution problem manually. However, such a pragmatic approach is tedious and error-prone. Without proper methods, techniques and tools to support evolution, model-driven engineering in general and domain-specific modelling specifically will not scale to industrial use.

1.1. Modelling Languages

To allow for a precise discussion of language evolution, we briefly introduce the concepts fundamental to modelling languages, in the context of multi-paradigm modelling [5].

The two main aspects of a model are its *syntax* (how it is represented) and its *semantics* (what it means).

Firstly, the syntax comprises *concrete syntax* and *abstract syntax*. The concrete syntax describes how the model is represented (in 2D vector graphical form for example), which can be used for model input as well as visualization. The abstract syntax contains the essence of the structure of the model (as an abstract syntax graph), which can be used as a basis for semantic anchoring [2]. A single abstract syntax may be represented by multiple concrete syntaxes. There exists a mapping between a concrete syntax and its abstract syntax, called the *parsing mapping function*. There is also an inverse mapping, called the *pretty printing mapping function*. Mappings are usually implemented, or can be at least represented, as model transformations.

[1] An early version of this work was previously presented at the Fujaba Days '09 workshop in Eindhoven, The Netherlands

Email addresses: Bart.Meyers@ua.ac.be (Bart Meyers), Hans.Vangheluwe@ua.ac.be (Hans Vangheluwe)

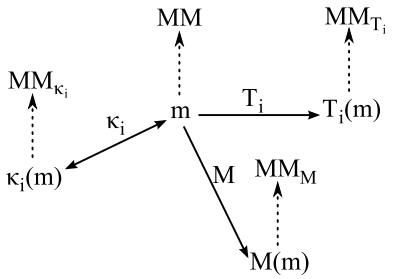

Figure 1: A model and its relations in MPM.

Secondly, the semantics of a model are defined by a complete, total and unique *semantic mapping function* which maps every model in a language onto an element in a *semantic domain*, such as differential equations, Petri Nets, or the set of all behaviour traces. Semantic mapping functions are performed on the abstract syntax for convenience.

A meta-model is the finite and explicit description of the abstract syntax of a language. Often, the concrete syntax is also described by (another) meta-model. Semantics are however not covered by the meta-model. The abstract syntax of the semantic domain itself will of course conform to a meta-model in its own right.

Figure 1 shows the different kinds of relations a model m is involved in. Relations are visualized by arrows, "conform to"-relationships are dotted arrows. The abstract syntax model m conforms to its meta-model MM. There is a bidirectional relationship κ_i (parsing mapping function and pretty printing mapping function) between m and a concrete syntax $\kappa_i(m)$. $\kappa_i(m)$ conforms to its meta-model MM_{κ_i}. Semantics are described by the semantic mapping function M, and map m to a model $M(m)$. $M(m)$ has syntax which conforms to MM_M. Additionally, there may be other transformations T_i defined for m.

2. Related Work

In order to be able to model evolution in-the-large, one should be able to model differences between two versions of a model [1, 13, 11, 18, 4]. Difference can be computed by traversing both graphs in parallel, and match either by using unique identifiers [1, 13], or by using a number of heuristics [11, 18]. These differences can then be represented as a so-called *delta model*. The difference can be represented by the edit operations that were performed on the model [1, 7] or by structurally decorating the metamodel with the changes through colouring [13, 18, 11, 14] or by using a designated meta-model that includes concepts of evolution [4, 15].

The most obvious side-effect of language evolution is the co-evolution of the instance models. This co-evolution is represented as a model transformation [19, 10, 15, 8, 6, 17, 16, 3, 7], which we will call the *migration transformation*. This migration transformation can be created manually [19, 8, 6], automatically [3] or a combination of both [17, 7].

3. Evolution for MPM

While model co-evolution as described above implements automation to some extent, there are other artifacts that might have to co-evolve. This section presents an exhaustive survey of possible evolutions and co-evolutions.

3.1. Syntactic Evolution

To get a general idea of the consequences of evolution, let us go back to Figure 1. When MM evolves, all models m have to co-evolve, which was discussed in Section 2. However, as the relations of Figure 1 suggest, the evolution of MM might affect other artifacts. First, similar to m, (the domain and/or image of) transformations such as κ_i, T_i and M might no longer conform to the new version of the metamodel. As a consequence, they too have to co-evolve. This makes all relations (syntactically) valid once again, which means that the system is syntactically *consistent* again. In short, meta-model evolutions can only be useful when both their model instances and related transformation models can co-evolve.

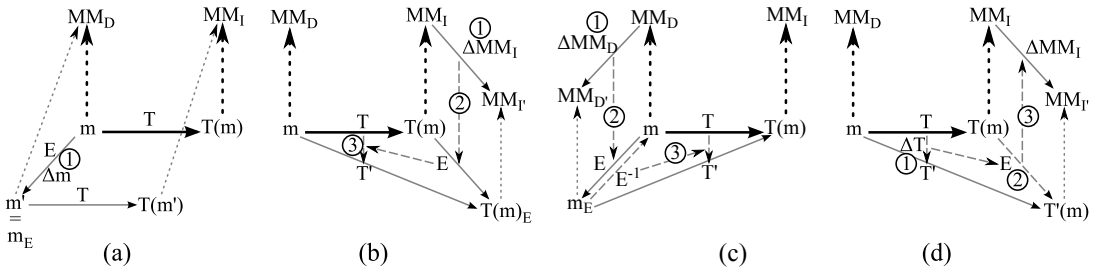

Figure 2: Co-evolution in (a) model evolution, (b) image evolution, (c) domain evolution and (d) transformation evolution.

However, there are more scenarios. Firstly, it is possible that the meta-model changes in such a way that the co-evolved models become structurally different, for example by removing a language construct. This means that each transformation defined for each co-evolved model has to be re-executed. The resulting co-evolved models can also be structurally different, so a chain of required evolution transformation executions may be required.

Secondly, changes made to one meta-model can reflect on another meta-model. For example, when a meta-element is added to a meta-model, a new meta-element is often also added to the meta-model of the concrete syntax(es) in order to be able to visualize this new construct. A similar effect can occur between any two related (by transformation) meta-models. In this sense, a chain of meta-model changes is again possible.

Thirdly, until now, we only discussed meta-model evolution as the driving force. Evolution of other artifacts, such as instance models and transformation models should also be taken into account. The case of the evolution of a model is trivial: related models can co-evolve by executing the respective transformations. Note however that a co-evolved model may be a meta-model, so that might trigger a number of co-evolutions of its own.

The case of the evolution of a transformation model can get complicated. In many cases though, the evolved transformation simply has to be executed again on each model it is defined for. However, this would restrict a transformation evolution to remain compliant to its source and target metamodels, which is not always what we want. For example, it might be possible that a new language is created by mapping rules for each language construct of an existing language. This is in particular convenient for creating a concrete syntax. On top of this, there are two additional special cases of transformation evolution. Firstly, the evolution of the parsing mapping function or the pretty printing mapping function requires the other one to co-evolve in order to maintain a meaningful relation between abstract and concrete syntax. Such a co-evolution can be generalized to any bidirectional transformation. Secondly, the evolution of the semantic mapping function requires a means to reason about semantics in order to trigger co-evolution, which brings us to the concept of semantic evolution.

3.2. Semantic Evolution

As mentioned above, semantics of a model are defined by its semantic mapping function to a semantic domain. Some analysis can be performed on models in this semantic domain (for example: check for a deadlock in a Petri Net). The results of this analysis can be considered a *property* of the model, or *P(m)*. A semantic mapping function is constructed in such a way that some properties $P_M(m)$ hold both for a model and for its image under the semantic mapping (i.e., the intersection of both property sets). These common properties have to be maintained throughout evolution. An evolution is a semantic evolution if some of these properties change. This typically happens when the requirements of a system change.

In general, when a model m in a formalism whose semantics is given by semantic mapping function M evolves to m', then $P_M(m')$ must be exactly $P_M(M(m))$ modulo the intended semantic changes. In general, when two versions of a system are (a) equal modulo their intended syntactic and semantic changes and (b) syntactically consistent, then the evolution of the system is *continuous*. Only continuous evolutions are deemed correct (and meaningful).

4. De-constructing Evolution

As discussed in the previous section, there are infinitely many possible co-evolution scenarios. Nevertheless, these scenarios can always be broken down into a few basic ones. Figure 2 shows the possibilities. Again, arrows

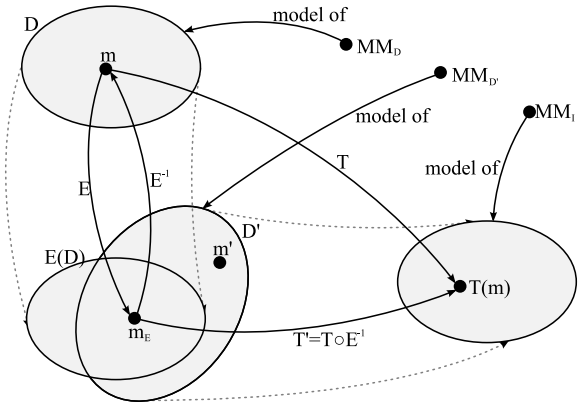

Figure 3: Domain-evolution as sets. The evolution *E(D)* does not map onto *D'* exactly. For *m'*, the constraint $T' = T \circ E^{-1}$ does not hold!

are transformations and dotted arrows are "conforms to"-relationships. Dashed arrows denote a (semi-)automatic generation. Each diagram starts from a bold relation between two meta-models MM_D and MM_I, modelled as a transformation T of models m.

4.1. Model Evolution

Figure 2 (a) shows model evolution. Some model m evolves to m'. In step 1 (the only step), a delta model Δm is constructed (either automatically or manually) that models the evolution of m to m'. This means that $m' = m + \Delta m$. The evolution itself is typically represented as a migration transformation, namely E. The equation $m_E = m + \Delta m = m'$ is valid. As previously discussed, because m evolved to m', every transformation T must be executed again, resulting in $T(m')$, conform to MM_i.

4.2. Image Evolution

Image evolution is shown in Figure 2 (b). Suppose that a meta-model MM_I evolves to $MM_{I'}$. In step 1 a delta model ΔMM_I is constructed to represent the difference between MM_I and $MM_{I'}$. In step 2 a migration transformation E is generated out of ΔMM_I. The execution of E co-evolves models $T(m)$ to $T(m)_E$, so that they conform to the new meta-model $MM_{I'}$. Moreover, the execution transformation T has to result in valid models (i.e., conform to $MM_{I'}$). As a consequence, T has to co-evolve to a new transformation T' (as in step 3), which is able to transform every possible m that conforms to MM_D, to $T(m)_E$. The diagram presents a solution for the generation of this T': for every m, $T'(m) = E(T(m))$ holds, or in short, $T' = E \circ T$. The co-evolution T' can be simply composed out of T and E.

4.3. Domain Evolution

Figure 2 (c) shows domain evolution, where MM_D evolves. The artifacts that co-evolve are similar to image evolution. This time however, T can be expressed as $T' = T \circ E^{-1}$. So, in this case, an inverse transformation E^{-1} needs to be constructed. Unfortunately, this equation does not hold for the entire domain D', as shown in Figure 3. The migration transformation E projects the entire domain D to $E(D)$, but it is possible that $E(D) \neq D'$. For m in Figure 3 it may be possible to construct E^{-1} such that $T'(m_E) = T(E^{-1}(m_E))$ holds. However, for m', which is an element of $D' \setminus E(D)$, this is not possible. Nevertheless, T' must apply to its entire domain D', so the equation $T' = T \circ E^{-1}$ can not be used for all possible models conform to $MM_{D'}$.

4.4. Transformation Evolution

Figure 2 (d) shows transformation evolution. The requirements of a system can change, resulting in the adjustment of the (desired) properties of a model. If transformations evolve according to a delta model ΔT, it is possible that they only have to be executed once again. In this case, the changes on the transformation are limited: the image of T' must conform to MM_i. As previously discussed, other artifacts might possible co-evolve. In this case, a migration transformation E must be composed from which a delta model ΔMM_i can be constructed.

4.5. Evolution Scenario Amalgamation

Using a combination of these four scenarios, all possible evolutions can be carried out. Note however that the problem of Figure 3 applies, so automated co-evolution is not always possible. The so-called unresolvable changes can be classified as models in $E(D) \setminus D'$. On the other hand, the transformation has to support the models in $D' \setminus E(D)$. We call this the *projection problem*. In general, the projection problem arises when $dom_E(T) \nsubseteq dom(T)$.

5. Conclusions

Widespread adoption of model-driven engineering is hampered by the lack of support for modelling language evolution. In domain-specific modelling in particular, modelling languages change frequently. When such languages evolve, support for (semi-)automated co-evolution of instance models is desirable.

In this paper, we have broken down this problem into four primitive (co-)evolution scenarios which can subsequently be combined. We showed that the co-evolution of transformations can be problematic, as a transformation always needs to be able to transform all possible elements in its domain. The presented breakdown serves as a starting point for further research into evolution solutions.

6. Bibliography

[1] Alanen, M., Porres, I., 2003. Difference and union of models.
URL http://citeseer.ist.psu.edu/596185.html

[2] Chen, K., Sztipanovits, J., Neema, S., 2005. Toward a semantic anchoring infrastructure for domain-specific modeling languages. In: EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software. ACM, New York, NY, USA, pp. 35–43.

[3] Cicchetti, A., Ruscio, D. D., Eramo, R., Pierantonio, A., 2008. Automating co-evolution in model-driven engineering. In: EDOC '08: Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference. IEEE Computer Society, Washington, DC, USA, pp. 222–231.

[4] Cicchetti, A., Ruscio, D. D., Pierantonio, A., 2007. A metamodel independent approach to difference representation. Journal of Object Technology 6 (9), 165–185.

[5] Giese, H., Levendovszky, T., Vangheluwe, H., October 2006. Summary of the workshop on multi-paradigm modeling: Concepts and tools. In: Kühne, T. (Ed.), Models in Software Engineering Workshops and Symposia at MoDELS 2006. Vol. 4364 of LNCS. Springer-Verlag, pp. 252–262.

[6] Gruschko, B., Kolovos, D., Paige, R., 2007. Towards synchronizing models with evolving metamodels. In: Proceedings of the International Workshop on Model-Driven Software Evolution at IEEE European Conference on Software Maintenance and Reengineering (ECSMR).

[7] Herrmannsdoerfer, M., Benz, S., Juergens, E., 2009. Cope - automating coupled evolution of metamodels and models. In: Proceedings of the 23rd European Conference on Object-Oriented Programming (ECOOP). pp. 52–76.

[8] Hoessler, J., Soden, J., Michael, Eichler, H., 2005. Coevolution of models, metamodels and transformations. Models and Human Reasoning, 129–154.

[9] Kelly, S., Tolvanen, J.-P., March 2008. Domain-Specific Modeling: Enabling Full Code Generation. John Wiley & Sons.
URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0470036664

[10] Lämmel, R., Nov. 2004. Coupled Software Transformations (Extended Abstract). In: First International Workshop on Software Evolution Transformations.

[11] Lin, Y., Gray, J., Jouault, F., 2007. Dsmdiff: A differentiation tool for domain-specific models. European Journal of Information Systems 16 (4, Special Issue on Model-Driven Systems Development), 349–361.
URL http://www.cis.uab.edu/gray/Pubs/ejis-2007.pdf

[12] Mosterman, P. J., Vangheluwe, H., 2004. Computer automated multi-paradigm modeling: An introduction. In: SIMULATION80. Vol. 9. pp. 433–450.

[13] Ohst, D., Welle, M., Kelter, U., 2003. Differences between versions of UML diagrams. SIGSOFT Softw. Eng. Notes 28 (5), 227–236.

[14] Schmidt, M., Gloetzner, T., 2008. Constructing difference tools for models using the sidiff framework. In: ICSE Companion '08: Companion of the 30th international conference on Software engineering. ACM, New York, NY, USA, pp. 947–948.

[15] Sprinkle, J., Karsai, G., April 2004. A domain-specific visual language for domain model evolution. Journal of Visual Languages and Computing 15.

[16] Vermolen, S., Visser, E., 2008. Heterogeneous coupled evolution of software languages. In: MoDELS '08: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems. Springer-Verlag, Berlin, Heidelberg, pp. 630–644.

[17] Wachsmuth, G., Jul. 2007. Metamodel adaptation and model co-adaptation. In: Ernst, E. (Ed.), Proceedings of the 21st European Conference on Object-Oriented Programming (ECOOP'07). Vol. 4609 of Lecture Notes in Computer Science. Springer-Verlag, pp. 600–624.

[18] Xing, Z., Stroulia, E., 2005. Umldiff: an algorithm for object-oriented design differencing. In: ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. ACM, New York, NY, USA, pp. 54–65.
URL http://dx.doi.org/10.1145/1101908.1101919

[19] Zhang, J., Gray, J., 11 2004. A generative approach to model interpreter evolution. In: OOPSLA Workshop on Domain-Specific Modeling. pp. 121–129, vancouver, VC.