

# A Framework for Transformation Chain Development Processes

Bert Vanhooff, Dhouha Ayed, and Yolande Berbers

Department of Computer Science, K.U. Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium

{bert.vanhooff, dhouha.ayed, yolande.berbers}@cs.kuleuven.be

**Abstract.** Model Driven Development (MDD) promotes the use of abstract models in software development. A key ingredient of MDD is the application of transformations to these models, which means that part of the development effort is relocated to the transformations. Currently there is almost no available guidance to help designing a suitable, project specific, transformation chain. We propose a framework of four concern layers to organize transformations, which facilitates better separation-of-concerns and offers opportunities for transformation reuse and replacement. We use this framework as a foundation to build an incremental transformation chain design process.

## 1 Introduction

Model Driven Development (MDD) is an approach to developing software that proposes using machine-readable models at various levels of abstraction as its main artifacts. The key MDD idea is to (semi-)automatically transform highly abstract models into more concrete models from which an implementation can straightforwardly be generated.

However, it is not enough to have a set of interesting development ideas and concepts to create great software; the way we use these is just as important. For example, it is not just because we implement an application using an object oriented language that our software is automatically well structured. The same is true for MDD: it is not because we apply an MDD approach that we get a good system. A good design of the system helps development just as much, if not more, as the development paradigm we use to realize it.

In this paper we argue that it is possible to achieve a better separation of concerns than with the classical PIM/PSM (Platform Independent/Specific Model) distinction (Section 2). In Section 3 we introduce a framework for transformation development processes that can be refined with concrete activities. We wrap up by presenting related work (Section 4), drawing some conclusion and discussing future work (Section 5).

## 2 Separation of Concerns with Transformations and Abstract Platforms

Many introductions on MDD use the notions of PIM and PSM, which were introduced by OMG's MDA [1]. A PIM is a model of a system that contains no technical details while a PSM is an elaboration of the same system that contains exactly these details. A single-level transformation process between PIM and PSM allows us to capitalize on stable platform independent matters and generate PSMs for a range of different concrete technology platforms.

The use of transformations can provide a more fine grained sense of separation-of-concerns than is implied by the black-and-white PIM/PSM separation. A *transformation chain* specifies how a number of transformations work together, each elaborating on the source model to come to a target model. The following types of concerns could subsequently be addressed in a transformation chain:

**Functional concerns** influence a model at the highest level of abstraction, namely the application business domain. For transformations this can mean automatic insertion of additional (pluggable) functionality into the basic model and can be accomplished with what is called model composition or weaving. – e.g. merge a basic chat application model with a model for providing file transfer.

**Non-functional concerns** determine the quality characteristics of a system. We consider them in a technology independent way at this stage. This means that transformations at this level can only apply generic solutions for a non-functional concern, without referring to platform-specific solutions. – e.g. a general distribution model, general persistence specification.

**Technical concerns** worry about the realization of the above (non-)functional concerns by using technology specific mechanisms (middleware concepts). – e.g. distribution realized with CORBA, persistency with an RDBM.

**Implementation concerns** go further than middleware-related concerns. Transformations for this category prepare a model for implementation in a certain programming language. – e.g. CORBA in Java, PG/SQL as RDBM.

The above categorization depicts a four-layered transformation approach where each layer can contain a number of concern-specific transformations that facilitate a gradual move from platform independency to platform specificity. The layers introduce constrained boundaries for transformations, forcing a narrower focus for each transformation and therefore taking one step towards easier reuse and replacement.

Each intermediate model can be seen as being specific to a virtual platform, corresponding the a concern layer, but independent to platforms further up the transformation chain. Abstract platforms are introduced in [2] and can serve as boundaries between transformations. A simple example of abstract platform is a user-defined subset of the UML with all semantic variation points fixed.

### 3 Transformation Chain Development Process

The discussion in the previous section provides a certain mind set to start thinking about transformations. Nevertheless this is often not enough to design a good transformation chain. We need some additional guidance that helps us decide which concrete models, notations and transformations are needed in the same way that classical software development processes, like the Unified Process (UP), offer guidance for classical application development starting from a set of requirements.

We expect a transformation chain design process to produce a model that has the following characteristics:

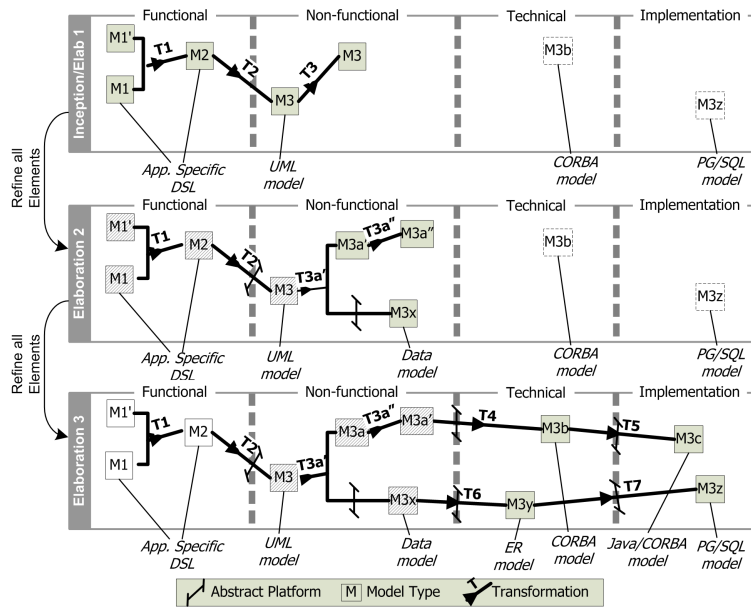
- Mutually exclusive transformations – Avoid overlap between transformation.
- Clearly separated transformation (concern) areas – A concern can easily be traced to a (group of) transformation(s).
- Loose coupling – Transformations do not rely on each other’s implementation properties but only on externally defined pre- and postconditions (reusability).
- Technology independence – The transformation chain model is specified independent of the technology that is used to implement it.

We present the outline of an iterative process framework that can act as a starting point for concrete transformation chain design processes based on the notion of the four-layer concern separation defined in Section 2. We name the iterations *inception* and *elaboration* corresponding to UP terminology. The evolution of a transformation chain model throughout the iterations is shown in Figure 1. Model types are shown as square boxes (labeled Mx, gray boxes are additions and hashed boxes are refinements), transformations are indicated as arrows (labeled Tx), abstract platforms are shown as solid lines and concern layers are shown as dotted lines. This example just illustrates the main concepts and is purely fictional.

#### ***Inception: MDD-specific Vision and Concerns***

A first iteration should clearly state a fair amount of the requirements that we impose on the transformation chain in order to get a common vision. Three important things have to be identified here:

1. Motivation for applying an MDD approach. This states the general quality requirements for the transformation chain and could be elaborated by using some form of transformation use cases. – e.g. offer a domain specific language (DSL) (top left of Figure 1), automate error-prone coding parts.
2. Concerns that you want to address in transformations instead of explicitly specifying them in the main application model – e.g. Distribution and persistence (both non-functional).
3. Pre-fixed constraints for the project that could influence the transformation chain development. – e.g. the use of CORBA and PG/SQL (the first is a technical concern, the second an implementation concern), using the UML as primary modeling language. Some of the constraints can already be indicated



**Fig. 1.** From top to bottom: incremental development of a transformation chain, showing the four concern layers with transformations, model types and abstract platforms.

on the transformation chain model (M3b and M3z on top of Figure 1 indicate the CORBA and PG/SQL constraints).

### *Elaboration 1: Conceptual Transformation Chain Model*

Since we aim for an incremental top-down approach the first elaboration iteration will primarily address the functional and non-functional layers. The goal here is to identify most of the transformations and model types in this area by describing them informally according to the concerns they address (e.g. persistency transformation, data model, distribution model, etc.) and decompose them according to common subconcerns.

In the top part of Figure 1 we show three transformations: the first one (T1) weaves two models in the application's DSL, T2 translates the DSL models into an equivalent UML representation and T3 represents the persistency transformation concern.

If we identify transformation overlap, e.g. we can have a 'notifier' concept from the persistence concern and an 'event' concept from a logging concern (not shown in the figure), this can be handled by a designated transformation addressing event and notification as separate subconcern. Hence we split T3 in T3a' and T3a'' (middle part of Figure 1). This kind of decomposition leads to more focused and potentially more reusable subject-matter specific transformations.

### ***Elaboration 2: Refined Transformation Chain Model***

In this iteration, we refine the conceptual model by formalizing the in- and output model types of each transformation and, accordingly, determine the abstract platforms.

We decided for example to let T3a' also output a dedicated data model besides a refined UML model (middle of Figure 1) – another decomposition. If in- and output of a transformation have a different model type, we have to add an platform boundary (between M3 and M3x). Each abstract platform is described by a new or existing/modified metamodel.

Finally we can describe the transformations in more detail by using the vocabulary defined in the metamodels/abstract platforms.

### ***Elaboration 3: Fully Specified Transformation Chain model***

In the last iteration (in practice there could be more) we execute further transformation decompositions and we consider the technical and implementation layer. Furthermore, the existing chain can be refined at all points in general.

If we consider transformations that operate within the same metamodel (or platform), some more fine-grained decompositions might be possible since each transformation is expressed in detailed metamodel/platform concepts instead of high-level, informal concern concepts (as in elaboration 1). A typical example in the UML domain is the addition of a separate transformation that generates get and set operations because these are UML specific concepts.

To close the gaps to the technical and implementation domain, we introduce for example the Entity-Relationship (ER) model (bottom of Figure 1) between the data model and the pre-determined PG/SQL model. Delaying these layers up till now, improves the separation between them and the higher layers. Furthermore the model types (metamodels) for the lower layers are assumed to be well-known since they correspond with concrete platforms.

## **4 Related Work**

Our work is partly inspired by [3], where the importance of preparation phases (development transformations, metamodels, model notations, etc.) in developing a project-specific MDD infrastructure is emphasized. We addressed one part of such an infrastructure: the transformation chain.

We argued for four conceptual concern layers, each in which many transformations can operate. The Enterprise Fondue method [4] introduces a layering that uses UML profiles to distinguish five abstraction layers: component, concern refinement, technical, platform and implementation. This method is aimed towards distribution and other middleware related concerns. Similarly the RM-ODP (Reference Model for Open Distributed Processing) [5] defines five viewpoints to look at a distributed system: enterprise, information, computational, engineering and technology. Our layering is very similar to the two previous ones but we wish to use it to address a more broad area of concerns. Almeida et al. introduced the notion of abstract platform [2] in particular for separating mid-

aware specific concerns and discusses the costs and benefits of more levels of abstraction in [6].

We did not discuss technical solutions for decomposing transformations, however some work has been done in this area. In [7] an OCL-based method is offered to formally specify pre- and postconditions for transformations, which is required to make transformations self-contained and easily reusable. In [8] we discussed a UML-based traceability mechanism that provides additional means to split up transformations into smaller, commonly used, pieces.

## 5 Conclusions and Future Work

An important part of the effort in an MDD-based project lies in the development of an appropriate transformation chain, which in turn eases the construction of the application(s) described in a project.

We introduced a four-layer concern framework for transformation chains that is richer than the classical PIM/PSM separation. To make good use of this framework we presented an initial high-level transformation chain design process that uses the layers to incrementally guide the designer to a complete transformation chain model. The layered approach facilitates a better separation-of-concerns and is a first step towards reusable and replaceable transformation components.

All ideas presented in this paper need to be refined. In our future work we will address concrete approaches for defining the layers formally technical solutions for composing transformations. We will also refine the proposed high-level process with concrete activities that offer better guidance to developers in order to evolve to a full-fledged transformation chain design process.

## References

1. Object Management Group: Mda guide version 1.0.1. Misc (2003)
2. Almeida, J.P., Dijkman, R.M., van Sinderen, M., Pires, L.F.: On the notion of abstract platform in mda development. In: EDOC. (2004) 253–263
3. Gavras, A., Belaunde, M., Almeida, L.F.: Towards an mda-based development methodology. In: EWSA. (2004) 230–240
4. Silaghi, R., Fondement, F., Strohmeier, A.: Towards an mda-oriented uml profile for distribution. In: EDOC. (2004) 227–239
5. ISO/IEC and ITU-T: Reference model for open distributed processing. (Misc)
6. Almeida, J.P.A., Pires, L.F., van Sinderen, M.: Costs and benefits of multiple levels of models in mda development. In: 2nd European Workshop on Model-Driven Architecture with Emphasis on Methodologies and Transformations. Volume Technical Report No. 17-04., University of Kent, Canterbury, UK (2004)
7. Cariou, E., Marvie, R., Seinturier, L., Duchien, L.: Ocl for the specification of model transformation contracts. In Patrascoiu, O., ed.: OCL and Model Driven Engineering, UML 2004 Conference Workshop, October 12, 2004, Lisbon, Portugal, University of Kent (2004) 69–83
8. Vanhooff, B., Berbers, Y.: Supporting modular transformation units with precise transformation traceability metadata. In: ECMDA Traceability Workshop, SINTEF, (2005) 15–27