Costs and Benefits of Multiple Levels of Models in MDA Development

João Paulo Almeida, Luís Ferreira Pires and Marten van Sinderen

Centre for Telematics and Information Technology, University of Twente
PO Box 217, 7500 AE Enschede, The Netherlands
{almeida, pires, sinderen}@cs.utwente.nl

Abstract. In Model-Driven Architecture (MDA) development, models of a distributed application are carefully defined so as to remain stable in face of changes in technology platforms. As we have argued previously in [1, 3], models in MDA can be organized into different levels of platform-independence. In this paper, we analyze the costs and benefits of maintaining separate levels of models with transformations between these levels. We argue that the number of levels of models and the degree of automation of transformations between these levels depend on a number of design goals to be balanced, including those of maximizing the efficiency of the design process and maximizing the reusability of models and transformations.

1 Introduction

The development of a distributed application can be regarded as the process of building a realization of the application that satisfies user requirements. In most traditional development cultures, application developers are instructed to produce intermediate models to facilitate bridging the gap between requirements and realization. These intermediate models are mainly regarded as a means to obtain a realization of the system, with different models addressing different design concerns. The ultimate product of the development process is the realization, which can be deployed on available implementation technologies (platforms). Any intermediate models produced during the development processes are considered means and not ends.

In the case of Model-Driven Architecture (MDA) development [8], however, intermediate models that are used to produce the final realization are also considered final products of the development process. These models are carefully defined so as to remain stable in face of changes in platform technologies, and are therefore called platform-independent models (PIMs).

In MDA development, models can be organized into different levels of platform-independence [1]. Models at a particular level of platform-independence can be realized into a number of platforms. When multiple levels of platform-independence are adopted, successive (automated) transformations may be used that lead ultimately to platform-specific models (i.e., models at the lowest level of platform-independence with respect to a particular definition of platform).

An indispensable activity in early stages of MDA development is to determine which levels of models and which (automated) transformations are necessary. This activity is part of the *preparation phase* of the MDA development process [4]. In the preparation phase, (MDA) experts define the metamodels, profiles and transformations that are to be used in the *execution phase* by application developers.

The organization of the execution phase in terms of levels of models depends on a number of design goals to be balanced, including those of maximizing the efficiency of the design process and maximizing the reusability of models and transformations. In this paper, we analyze the factors that should be considered in order to determine the organization of the execution phase. We claim no conclusive or concrete guidelines on the use of different levels of models and transformations. We rather aim at setting the stage for further discussion on this very important issue for MDA development.

The concept of abstract platform we have proposed in [1, 3] supports us in the discussion. An abstract platform is an abstraction of infrastructure characteristics assumed for models of an application at a certain level of platform-independence. An abstract platform is represented through metamodels, profiles and reusable design artifacts [3]. For example, if a platform-independent design contains application parts that interact through operation invocations (e.g., in UML [10]), then operation invocation is a characteristic of the abstract platform. Capabilities of a concrete platform are used during platform-specific realization to support this characteristic of the abstract platform. For example, if CORBA [5] is selected as a target platform, this characteristic can be mapped onto CORBA operation invocations.

This paper is further structured as follows: section 2 discusses how the automation of transformations between two levels of models can be justified; section 3 considers the use of intermediate levels of models, and section 4 provides some concluding remarks.

2 Introducing Automated Transformations

During the execution phase of an MDA project, an application developer derives models at a lower-level of platform independence from models at a higher-level of platform independence. In order to increase the efficiency of the application development process, the developer may use automated transformations to bridge between different levels of models.

A requirement to the automation of transformation is the specification of transformation in the preparation phase. Full automation of transformation between two levels of models requires the transformation specifier to define rules to transform all possible source models into appropriate target models. The transformation specifier must fully understand the relation between source and target metamodels, and express these rules in a suitable transformation language, supported by a transformation tool. For these reasons, transformation specifications should be produced by a knowledgeable (MDA) expert.

When transformation is automated, the creative design activities that would normally be executed manually by a designer are generalized and moved to the specification of the transformation itself and to the application of marks (marking). The costs of defining an automated transformation between two related levels of models *A* and *B* must be compensated by reusing the transformation. The following conditions contribute to the reuse of the transformation:

- *the number of applications built using models at level A and targeting B is high*, i.e., the (abstract) platform at level *B* is popular for targeting applications that can be expressed in terms of (abstract) platform at level *A*;
- *changes in application requirements are frequent*, but these changes do not affect the stability of the (abstract) platform at level *A*;
- *the development process is cyclic, and the number of design iterations is high*, i.e., the model of the application in *A* is modified several times during the development. In this case, manual manipulation of models would have required manual propagation of changes applied at level *A*.

The bottom-line is that the cost of building an automated transformation between levels *A* and *B* must be lower than the costs of manually deriving models at level *B* (from designs at level *A*) over (a long period of) time. Therefore, the stability of the (abstract) platforms at level *A* and *B* should be considered. The stability of the (abstract) platform at level *A* allows more applications to be developed in terms of this platform and the stability of (abstract) platform at level *B* is required to reuse the transformation, since transformation from *A* to *B* is specific to the platform at level *B*.

It is possible that models obtained manually and automatically differ significantly with respect to relevant qualities. These qualities should be considered when justifying automation. For example, depending on the transformation, automated code generation may result in implementations of lower time performance. When this is the case, this can be reflected in cost estimates by lowering the cost of manual coding to account for the benefits of obtaining implementations that perform better. Automated code generation may also lead to improving the correctness of implementations. In this case, cost estimates should include the costs incurred by testing, both for testing the transformation and testing the code obtained manually.

3 Introducing Intermediate Levels of Models

We envision two different extreme approaches to organizing the development process with respect to platform-independence levels:

i. *an approach with minimal use of levels of platform-independence*, in which one level of platform-independent models and one level of platform-specific models are related (through a fully or partially automated transformation), and;
ii. *an approach with exhaustive use of intermediate platform-independence levels* and several (fully or partially automated) transformations between these models.

We argue that a combination of these extreme approaches is the most effective way to handle the problem. In the sequence, we consider the costs and benefits of introducing an intermediate level of models between two arbitrary levels, a source level and a target level. This allows us to consider the full range of combinations of the extreme approaches (i) and (ii), since the recursive introduction of intermediate levels eventually leads to an exhaustive use of intermediate levels. In the discussion, we distinguish between fully or partially automated transformations.

3.1 Fully automated transformations

Figure 1 depicts the alternative situations which we contrast for fully automated transformations: (a) a situation in which a transformation *T* produces models at level *B* from models at level *A*, and (b) a situation in which a transformation T_1 produces models at level *X* from models at level *A*, and a transformation T_2 produces models at level *B* from models at the intermediate level *X*. The arrows in Figure 1 represent the execution of a transformation.

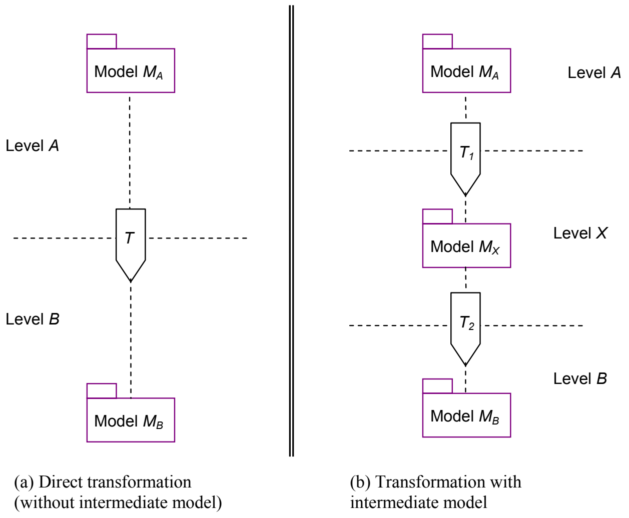

Fig. 1. Direct transformation and transformation with intermediate model

Considering solely the effort spent in the preparation phase to specify the transformations in situations (a) and (b), we cannot formulate a general rule to decide whether an intermediate step should be introduced. In some cases, it may be easier to define two transformations using an intermediate model, and, in some other cases, direct transformations may be easier to define.

Nevertheless, it is possible to draw some general conclusions on the consequences of introducing intermediate levels of models for the reuse of transformations. In this respect, an intermediate level of models may be beneficial since:

1. it may be possible to *reuse the transformation from source models to intermediate models*, even if the original transformation from intermediate models to target models cannot be reused (e.g., because of platform change); and,
2. it may be possible to *reuse the transformation from intermediate models to target models* in new projects, since there may be transformations from different source levels to the intermediate level.

A transformation between levels *A* and *B* is specific to the (abstract) platform of level *B*. Therefore, the stability of (abstract) platform at level *B* is required to reuse the transformation. Introducing an intermediate level of models may serve to factor out parts of the transformation that are less platform-specific, capturing unstable transformation *X* to *B* separately from stable transformation *A* to *X*. For example, consider that the level *A* consists of models in an application-domain-specific language [2], and that level *B* consists of middleware platforms, such as CORBA/CCM [5, 9] and Web Services [14, 15]. Instead of defining a transformation directly from *A* to *B*, one may consider the introduction of EDOC CCA models [11] as intermediate models at level *X*, capturing a transformation from the domain-specific language to a solution that is more stable than middleware platforms. Additional transformations that do not have to consider the specificities of the domain-specific language can be used to transform the EDOC CCA models to CORBA/CCM or Web Services PSMs. Clearly, this solution requires the stability of the intermediate level *X*, in the example, EDOC CCA models. This solution is depicted in Figure 2(a).

A transformation between levels *A* and *B* is also specific to the (abstract) platform of the source level *A*. Introducing an intermediate level of models may also lead to the reuse of the transformation from the intermediate model to the target model. For example, consider that the level *A* consists of models in different application-domain-specific languages, and level *B* consists of Web Services. Introducing an intermediate level *X*, e.g., populated with EDOC CCA models allows us to reuse the general-purpose EDOC to Web Services transformation. This transformation is not "contaminated" with application-domain-specific issues. Again, this solution requires the stability of the intermediate level *X*. This solution is depicted in Figure 2(b). While we have presented the two solutions separately, they could be combined, as depicted in Figure 2 (c).

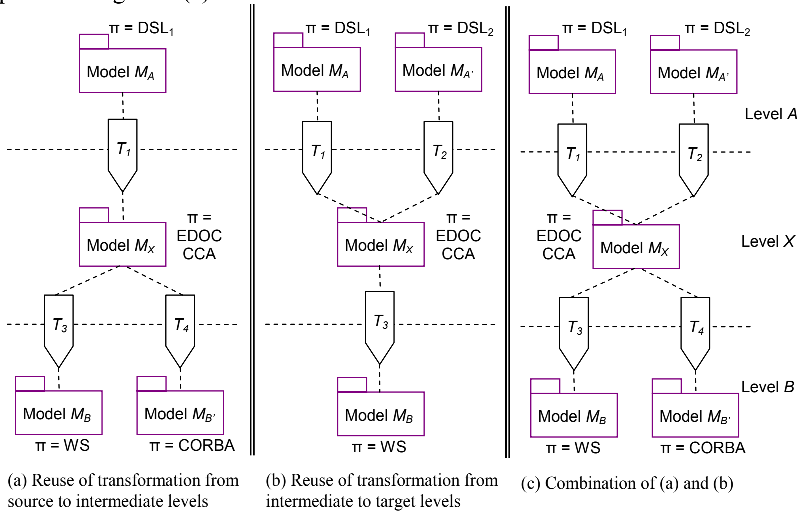

Fig. 2. Reuse of transformations due to introduction of intermediate level of models

In order to justify the introduction of the intermediate levels of models *X*, the abstract platform of the level *X* must be suitable for a large number of applications that can be described at level *A* and realized on platforms at level *B*. In our example, the consequence of this observation is that the abstract platform at level *X* should be independent of application domains at level *A* and independent of technology platforms at level *B*. In addition, standardization of this abstract platform may be necessary to increase the opportunities for the reuse of transformations to and from the intermediate level. The EDOC CCA is an example of such an abstract platform, allowing the description of distributed application in terms of components and their interconnection in terms of messages exchanged through ports.

The same pattern of transformation reusability can be observed when considering the transformation of EDOC CCA models at level *X* to models at the level of programming languages such as Java. In this case, level *B* in Figure 2 can be regarded as an intermediate level in the transformation, consisting of CORBA and Web Services-specific models. These models are transformed into Java interfaces, stubs and skeletons through standardized transformations [7, 12]. These transformations are executed through tools such as the one available in [13] and the ones listed in [6].

3.2 Partially automated transformations

It may be necessary to introduce an intermediate level of models between a source and a target level when no automated transformation can be defined directly, or when automated transformations produce results that do not satisfy non-functional requirements. By introducing an intermediate level of models, intermediate models can be elaborated upon, e.g., incremented, modified, combined with additional models and marked. The intermediate level can be regarded as a means to systematically lowering the degree of automation, and introducing opportunities to insert design decisions in the transformation from source to target models.

For example, let us consider again level *A* consisting of models in application-domain-specific languages, level *X* consisting of EDOC CCA models and level *B* consisting of CORBA/CCM and Web Services-specific models. This situation is depicted in Figure 3. In this example, marking EDOC CCA models manually is a means to specify properties that are not stated in source nor intermediate models and that may be required for the realization of the application on a target middleware platform. These properties may be requirements on the replication of components to satisfy availability requirements, requirements on the potential location of components in the distributed environment to satisfy time performance requirements, requirements on the persistency mechanisms required, etc. These requirements refer to specific components in the EDOC CCA models and cannot be specified meaningfully at level *A* or derived directly from EDOC CCA models.

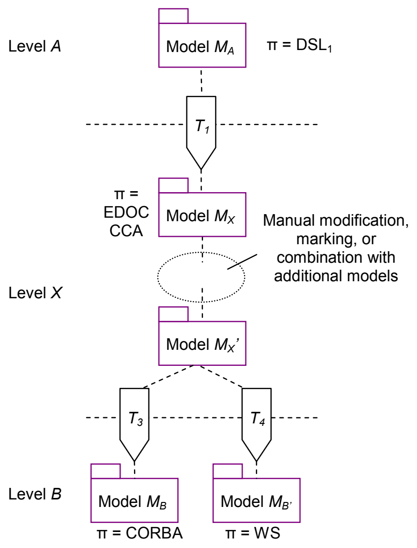

Fig. 3. Intermediate models as means to introduce design decisions

Reducing the level of automation of transformations incur additional costs on the introduction of an intermediate level of models. Changes in models at a high-level of platform-independence may lead to changes in all intermediate models and their associated markings. If intermediate models affected by changes need to be modified or marked manually, propagation of changes may lead to high costs. In contrast, in fully automated transformation chains, changes are automatically propagated through transformation. Since the state-of-the-art still requires significant developer intervention along transformation chains, the propagation of changes contributes to a large portion of the costs incurred by introducing separate levels of models. These costs should ideally be contained by appropriate traceability mechanisms in MDA tools.

With the introduction of an intermediate level of models, it may be necessary to develop specific languages, metamodels, profiles or marking models for that level. This incurs some additional effort for the preparation activities. For the case of partially automated transformation, developers using the intermediate models in execution activities must learn how to use the specific metamodels, profiles, or marking models required at that level, which usually incurs training costs and increases the threshold for developers to apply the particular model-driven development process.

4 Concluding remarks

In MDA development, opportunities for reuse of transformations play an important role in deciding the organization of the execution phase in terms of levels of models and transformations. A single transformation from high-level models to implementations may be costly to develop and is rendered useless in the face of technology platform changes. Given that technology platforms are generally regarded as unstable, it is important to attempt to recognize (intermediate) stable abstract platforms that can be used for a large number of applications. This allows transformations to and from this intermediate abstract platform to be reused.

In the example we have presented, we have considered an intermediate level of models based on the EDOC CCA UML profile, which enables the modeling of distributed applications as recursive compositions of abstract components. Recently, similar modeling capabilities have been incorporated in UML 2.0, with the introduction of composite structures [10]. Consequently, UML 2.0 and the EDOC CCA Profile can be seen as alternatives for modeling distributed applications. The proliferation of different (incompatible) intermediate levels of models reduces the opportunities for large-scale reuse of intermediate models and transformations to and from intermediate models. This calls for the standardization of a small number of abstract platforms that are, to a great extent, application-domain-neutral and platform-independent.

A conclusive study with respect to the costs and benefits of introducing different levels of models requires empirical verification. Such a study should consider a multitude of application requirements, as well as the opportunities for reuse across different instances of model-driven development projects.

In the absence of such an empirical study, we have discussed, in general terms, the benefits and costs of introducing different levels of models and transformations. We believe this forms a basis to enable trade-off analysis between the different factors in the preparation phase of MDA development.

Evaluating these trade-offs at early stages of development remains nevertheless a challenging activity, since the benefits of the separation PIM/PSM must be considered on the long run, particularly due to the role of reuse of models and transformations. Important open issues are how to estimate the stability of concrete platforms, application domains and applications and how to define stable abstract platforms that should be standardized.

Acknowledgements

This work is part of the Freeband A-MUSE project. Freeband (http://www.freeband.nl) is sponsored by the Dutch government under contract BSIK 03025. This work is also partly supported by the European Commission within the MODA-TEL IST project (http://www.modatel.org).

References

1. Almeida, J.P.A., van Sinderen, M., Ferreira Pires, L., Quartel, D.: A systematic approach to platform-independent design based on the service concept. In: Proceedings 7th IEEE Intl. Enterprise Distributed Object Computing Conference (EDOC 2003), IEEE Computer Society, Los Alamitos, CA (Sept. 2003) 112–123
2. van Deursen, A., Klint, P., and Visser, J.: Domain-Specific Languages: An Annotated Bibliography. In: ACM SIGPLAN Notices 35(6), (June 2000) 26–36
3. Almeida, J.P.A., Dijkman, R., van Sinderen, M., Ferreira Pires, L.: On the Notion of Abstract Platform in MDA Development. In: Proceedings 8th IEEE Intl. Enterprise Distributed Object Computing Conference (EDOC 2004), IEEE Computer Society, Los Alamitos, CA (to appear)
4. Gavras, A., Belaunde, M., Ferreira Pires, L., Almeida, J.P.A.: Towards an MDA-based development methodology for distributed applications. In: Proceedings of the 1st European Workshop on Model-Driven Architecture with Emphasis on Industrial Applications (MDA-IA 2004), CTIT Technical Report TR-CTIT-04-12, University of Twente, ISSN 1381 - 3625, Enschede, The Netherlands (March 2004) 43–51
5. Object Management Group: Common Object Request Broker Architecture: Core Specification, Version 3.0, formal/02-12-06 (2002)
6. Object Management Group: Getting Specs and Products, available at http://www.omg.org/gettingstarted/specsandprods.htm#GetProds
7. Object Management Group: IDL to Java Language Mapping, v1.2, formal/02-08-05 (2002).
8. Object Management Group: MDA-Guide, v1.0.1, omg/03-06-01 (June 2003)
9. Object Management Group: CORBA Component Model, Version 3.0, formal/02-06-65 (2002)
10. Object Management Group: UML 2.0 Superstructure, ptc/03-08-02 (2003)
11. Object Management Group: UML Profile for Enterprise Distributed Object Computing Specification, ptc/02-02-05 (2002)
12. Sun Microsystems, Inc.: JSR-000224 Java API for XML-Based RPC 2.0 (June 2003).
13. Sun Microsystems, Inc.: Java Web Services Developer Pack (Java WSDP), available at http://java.sun.com/webservices/jwsdp/index.jsp
14. World Wide Web Consortium: SOAP Version 1.2 Part 1: Messaging Framework, W3C Proposed Recommendation (2003), available at http://www.w3.org/TR/soap12-part1
15. World Wide Web Consortium: Web Services Description Language (WSDL) 1.1, W3C Note (2001), available at http://www.w3.org/TR/wsdl