Dependencies between Models in the Model-driven Design of Distributed Applications[1]

João Paulo A. Almeida, Luís Ferreira Pires, 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 our previous work, we have defined a model-driven design approach based on the organization of models of a distributed application according to different levels of platform-independence. In our approach, the design process is structured into a preparation and an execution phase. In the preparation phase, (abstract) platforms and transformation specifications are defined. These results are used by a designer in the execution phase to develop a specific application. In this paper, we analyse the dependencies between the various types of models used in our design approach, including platform-independent and platform-specific models of the application, abstract platforms, transformation specifications and transformation parameter values. We consider models as modules and employ a technique to visualize modularity which uses Design Structure Matrices (DSMs). This analysis leads to requirements for the various types of models and directives for the design process which reduce undesirable dependencies between models.

1 Introduction

In our previous work [1, 2], we have defined a model-driven design approach (aligned with the Model-Driven-Architecture [7]) based on the organization of models of a distributed application according to different levels of platform-independence. In this approach, models at a particular level of platform-independence can be realized with a number of platforms (such as, e.g., middleware platforms), possibly through application of successive (automated) transformations 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 important architectural concept of our approach is that of an abstract platform. 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 artefacts [1]. For example, if a platform-independent design contains application parts that interact through operation invocations (e.g., in UML [8]), then operation invocation is a characteristic of the abstract platform. Capabilities of a concrete platform are used

[1] 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.

during platform-specific realization to support this characteristic of the abstract platform. For example, if CORBA is selected as a target platform, this characteristic can be mapped onto CORBA operation invocations.

An indispensable activity in early stages of our development approach is to determine the levels of models, the abstract platforms, and the (automated) transformations that are needed. This activity is part of the *preparation phase* of the MDA development process [6]. In the preparation phase, (MDA) experts define the metamodels, profiles and transformations that are to be used in the *execution phase* by application developers. In the execution phase, a specific application is developed using the generalized designs and design knowledge captured during the preparation phase.

Figure 1 shows the various models manipulated in our approach. Three levels of platform-independence are depicted, and the results are classified according to the phase in which they are produced. In this figure, an arrow indicates that a model is dependent on the existence of another model by construction. Abstract platforms have been depicted as models, indicating that abstract platform definitions can be captured in *abstract platform models*. Transformation specifications have also been depicted as models, indicating that generalized design operations can be captured and reused. Transformation specifications can be parameterized and values for transformation parameters are defined in the execution phase. These values are called transformation arguments. Arguments of a transformation are also called *markings* when these are associated to elements in a source model, in which case transformation parameters are called *marks*.

Ideally, models in our approach (presented in Figure 1) should be independent of each other, i.e., it should be possible to create models independently, and a modification in one model should not impact other models. Nevertheless, models capture design decisions on the same object of design, i.e., the same application, and hence not all models are independent of each other. The benefits of separation of models are reduced when models are related in such a way that modifications in a model affect other models. In this paper, we analyse the dependencies between the various types of models used in our design approach and strive to find techniques to avoid undesirable dependencies between models.

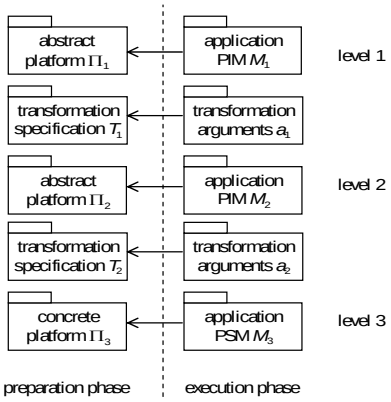

Fig. 1. Models in our design approach

Dependencies between models restrict the opportunities for division of labour and concurrent design. Interdependencies reduce the efficiency of the design process and often have to be addressed in the design process by introducing iteration cycles [4]. As we elaborate in this paper, some interdependencies can be avoided by following a number of rules with respect to the content of the various models and with respect to the modifications that may be applied to the various models.

In the remainder of this paper, we address the following questions with respect to the separation of models in our approach (among others):

- can concrete platforms be modified without affecting PIMs and abstract platforms?
- can transformation specifications be modified without affecting PIMs and abstract platforms?
- does a modification in a PIM affects a corresponding PSM?
- does a modification in a PSM affects a corresponding PIM?
- are there interdependencies between the various models that require iterations in the design process? Can these be avoided?

This paper is further organised as follows: section 2 proposes that models should be considered as modules whose modularity can be analysed through a technique called Design Structure Matrices (DSMs) [9, 10]; section 3 analyses the (inter)dependencies between the various types of models, which results in requirements and guidelines for the separation of models; section 4 discusses how the dependencies between models affect the design process; section 5 classifies the different models according to their various dependencies; finally, section 6 presents some concluding remarks.

2 Models as modules

In order to examine the relations between the various models, we consider models as *modules*. Typically, a module is a set of elements of a design that are grouped together according to an architecture or plan, with three main purposes [3, 4]: to make complexity manageable; to enable parallel work; and to accommodate future uncertainty.

While modularization is often used as a technique to split up and assign different functions of a complex system to different system parts, we split up and assign different design decisions to different models. A number of basic principles of modularity apply both to the functional decomposition of system parts (within a model) and to the separation of models in our design approach.

As is noted in [4]: “a complex engineering system is modular-in-design if (an only if) the *process of designing it can be split up and distributed across different separate modules* that are coordinated by design rules, not by ongoing consultations amongst the designers.” This definition reveals two important features of systems that are modular-in-design:

- *Independence*: The absence of ongoing consultations amongst the designers of different modules reveals that modules should be largely independent of each other. Modules correspond to independent activities in the design process; and

– *Dependence*: The relations between the different modules are defined by a set of design rules[2] to be respected. These design rules reflect the need for coordination of design choices. Separating strongly related modules forces the number of design rules to increase, constraining the freedom of designers of the different modules.

In the following sections, we examine independence and dependence of models in our design approach. We employ a technique to visualize modularity-in-design which uses Design Structure Matrices (DSMs) [9, 10]. DSMs have been used extensively in the field of Engineering Design, both for products and production processes and design processes [4]. In this technique, modules are arrayed along the rows and columns of a square matrix. The matrix is filled in by determining, for each module, which other modules affect it and which are affected by it. The result is a map of the dependencies between the modules.

3 Dependencies between models: two levels of models

We start our analysis by assuming two levels of design within a single design iteration cycle as depicted within the rounded rectangle in Figure 2.

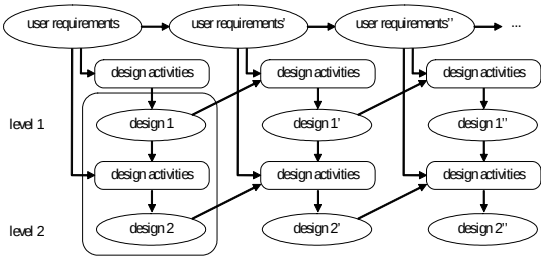

Fig. 2. Two levels of models related by transformation

We assume further that the preparation phase results in an abstract platform Π_1 for designs at level 1, a concrete platform Π_2 for designs at level 2. The design activities are constrained by a transformation specification T_1 that relates models that rely on Π_1 to models that rely on Π_2. This situation is depicted in Figure 3. This figure reveals the various models of the execution phase that are considered at this point of our analysis, namely, an application PIM, transformation arguments, and an application PSM.

[2] In functional decomposition, interfaces between components are considered design rules.

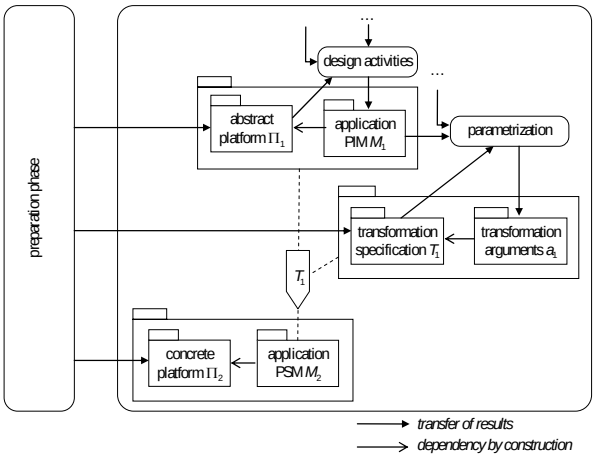

Fig. 3. Two levels of models related by transformation

We discuss the dependencies between each of the models depicted in Figure 3 in the following sections. In each section, we discuss how the various models are affected as a result of a modification of one of the other models. After the relations between all models are examined, a DSM is built to visualize the dependencies between the various models.

Application PIM. Table 1 shows the dependencies between the various models and an application PIM. The '⊠' symbol marks the existence of some dependency. The absence of the symbol indicates there is no dependency. We justify the existence or absence of a dependency for each pair of models.

Table 1. Dependencies between the various models and an application PIM

	Application PIM	Explanation
Application PIM	N/A	trivial
Abstract platform		An abstract platform is designed so that it can be used to design a class of applications; the modified application PIM is still a member of this class of applications. This constitutes a generality requirement for abstract platform, but also sets the constraints on possible modifications of an application PIM for a given abstract platform.
Application PSM	☒ through transformation	The relations between application PIMs and PSMs are determined by transformation specifications and transformation arguments; if the application PIM is modified, it is possible that the modified PIM and the original PSM no longer respect this relation; in this case, the PSM or transformation arguments may be affected by change.
Concrete platform		The concrete platform is a member of the set of target platforms implied by portability requirements; all application PIMs that rely on the abstract platform must be buildable (see explanation below about buildability) in the concrete platform, thus requiring no modifications in the concrete platform. This constitutes requirements for the abstract platform and transformation specification.
Transf. arguments	☒	Transformation arguments are used to introduce variation in transformation specifications, in order to capture particular design decisions; these decisions may be application-specific or may refer to elements of the application PIM; e.g., transformation parameters can be used to specify the physical allocation of each application component in the application PIM.
Transf. specification		Transformation specifications are designed so that they can be applied to the class of applications that can be built on top of an abstract platform; the modified PIM is still a member of this class of applications. This constitutes a generality requirement for transformation specification.

Buildability of a design is inversely proportional to the amount of time, effort and resources required to build a conformant realization of the design on *a particular platform*. Buildability depends on the contents of a design. The actual contents of a platform-independent design depend partly on the abstract platform, which is defined in the preparation phase. Therefore, in the preparation phase, buildability can only be estimated indirectly, by analysing the impact of abstract platform characteristics in the buildability of the class of application designs supported by the abstract platform. We propose this is done by examining the differences and similarities in the abstract platform and target platforms[3].

Having introduced the notion of buildability, we are able to formulate a definition of platform-independence of a design. We say that a design is *platform-independent* if, and only if, it is buildable on a number of target platforms. The set of target platforms is determined by *portability requirements* for the design, which are themselves determined by technical, business and strategic arguments.

Abstract platform. Table 2 shows the dependencies between the various models and an abstract platform.

[3] We have explored this idea initially in [2].

Table 2. Dependencies between the various models and an abstract platform

	Abstract platform	Explanation
Application PIM	☒	By definition: “an abstract platform is an abstraction of infrastructure characteristics assumed in the construction of PIMs of an application”; if these characteristics change, the application PIM may be affected.
Abstract platform	N/A	trivial
Application PSM		Modifying an abstract platform may affect PIMs, transformation specifications (see respective cells in this table), which in turn may affect application PSMs (see other tables); however, only direct dependencies are represented in a DSM.
Concrete platform		The set of target platforms is determined by portability requirements; during abstract platform definition, buildability with respect to the target platform must be observed. This constitutes a requirement for abstract platform definition.
Transf. arguments		Transformation arguments depend on the transformation specification, which depends on abstract platforms (see cell below); however, only direct dependencies are represented.
Transf. specification	☒	The abstract platform defines the common characteristics of a class of platform-independent designs for which there should be generalized implementation relations to different platforms; these implementation relations are captured in transformation specifications; a change in abstract platform characteristics changes the class of applications, invalidating assumptions on common concepts, patterns and structures that were made to define transformations.

The separation between an abstract platform and a transformation specification is analogous to the separation between an interface definition and a realization of the interface in component-based design: an abstract platform defines requirements which are satisfied by one or several transformation specifications.

Application PSM. Table 3 shows the dependencies between the various models and an application PSM.

Table 3. Dependencies between the various models and an application PSM

	Application PSM	Explanation
Application PIM	☒ through transformation	The relations between application PIMs and application PSMs are determined by transformation specifications and transformation arguments; if the application PSM is modified, it is possible that the modified PSM and the original PIM no longer respect this relation; in this case, the PIM or transformation arguments may be affected by change. This dependency exists for both unidirectional and bidirectional [5] transformations. In the case of bidirectional transformations, changes to PIM may be propagated automatically.
Abstract platform		A modification in an application PSM may result in a modification in the application PIM (see cell *application PIM* above); the modified PIM is still a member of this class of applications for which the abstract platform is defined. This constitutes a generality requirement for abstract platform, but also sets the constraints on modifications of an application PSM for a given abstract platform.
Application PSM	N/A	trivial
Concrete platform		A concrete platform is designed so that is can be used to design a class of applications; the modified PSM is still a member of this class of applications. This constitutes a generality requirement for concrete platforms.
Transf. arguments	☒ through transformation	(see cell *application PIM* above)
Transf. specification		Transformation specifications define generalized implementation relations; transformation specifications define a class of PSMs that conform with PIMs; the modified PSM is still a member of this class of applications. This constitutes a generality requirement for transformation specifications, but also sets the constraints on possible modifications of an application PSM for a given transformation specification and a PIM.

Concrete platform. Table 4 shows the dependencies between the various models and a concrete platform.

Table 4. Dependencies between the various models and a concrete platform

	Concrete platform	Explanation
Application PIM	independence is engineered	Independence is engineered in the definition of abstract platforms. This constitutes a buildability requirement for abstract platforms.
Abstract platform	independence is engineered	Independence is engineered in the definition of abstract platforms. This constitutes a buildability requirement for abstract platforms.
Application PSM	☒	Application PSM depends on sets of concepts, patterns and structures provided by a concrete platform; the instability of concrete platforms, and hence application PSMs, motivates separation of platform-independent and platform-specific concerns in our approach.
Concrete platform	N/A	trivial
Transf. arguments	☒	Transformation arguments may be platform-specific, e.g., markings may define that particular components should be transformed into Session or Message-Driven Enterprise Java Beans.
Transf. specification	☒	Transformation specifications define generalized implementation relations for a particular target platform; change the target platform and these relations may be invalidated. Ideally, this dependency could be reduced by using concrete platform models as transformation arguments. However, this solution requires highly general transformation specifications, which define generalized implementation relations for a class of target platforms (resulting in a platform-independent transformation specification).

Transformation arguments. Table 5 shows the dependencies between the various models and transformation arguments.

Table 5. Dependencies between the various models and transformation arguments

	Transf. arguments	Explanation
Application PIM		Abstract platforms are defined to preserve freedom of implementation, so that different implementations of application PIMs built on top of it are possible; since transformation arguments are used to introduce variations in generalized implementation relations, changes in transformation arguments should not affect application PIMs nor abstract platforms. This constitutes a requirement for abstract platforms and transformations, and sets the constraints on possible modifications of transformation arguments for a given combination of abstract platform and transformation specification.
Abstract platform		(see cell *application PIM* above)
Application PSM	☒ through transformation	The relations between PIMs, transformation arguments and PSMs are determined by transformation specifications; if transformation arguments are modified, it is possible that the original PIM, the modified arguments and the original PSM no longer respect this relation; in this case, the PSM may be affected by change in transformation arguments.
Concrete platform		A concrete platform is designed so that is can support a class of applications; a PSM that is affected by a change in transformation arguments is still a member of this class of supported applications, therefore, requiring no modification of the concrete platform. This constitutes a requirement for transformation specification, namely that the results of transformations are always PSMs that use the concrete platform.
Transf. arguments	N/A	trivial
Transf. specification		Transformation specifications have transformation parameters, which are assigned values when the transformation specification is instantiated.

From the perspective of model transformation, the distinction between PIMs and transformation arguments is unnecessary: both PIMs and transformation arguments may be considered as input information for an unparameterized transformation. However, the distinction is relevant from the perspective of the design process: PIMs are *platform*- and *transformation independent*, while transformation arguments may be *platform*- and *transformation specific*. Transformation arguments may be defined after PIMs have been conceived. As a consequence, designers of PIMs may not be aware of whatever transformation parameters may be chosen by a designer using the PIM as a starting point to derive a PSM.

Transformation specification. Finally, Table 6 shows the dependencies between the various models and a transformation specification.

Table 6. Dependencies between the various models and a transformation specification

	Transf. specification	Explanation
Application PIM		Abstract platforms are defined to preserve freedom of implementation, so that different implementations of application PIMs built on top of it are possible; these different implementations are captured in transformation specifications. This constitutes a requirement for abstract platform, but also sets the constraints on possible modifications of transformation specifications for a given abstract platform.
Abstract platform		(see cell *application PIM* above)
Application PSM	☒	The relation between application PIM and application PSM is determined by transformation specifications and transformation arguments; since a change in transformation specification should not affect PIMs (see cell *application PIM* above), modifications to transformation specifications must be accommodated in the PSM or in transformation arguments.
Concrete platform		PSMs related by transformation specifications must be realizable on top of a concrete platform. This constitutes a requirement for transformation specifications.
Transf. arguments	☒	Transformation parameters are used to introduce variations in generalized implementation specifications; if a transformation specification is modified parameters may be modified and new parameters may be introduced, affecting transformation arguments.
Transf. specification	N/A	trivial

Since transformation arguments may be transformation-specific, transformation arguments must be captured separately from PIMs so that PIMs do not become transformation-specific. Therefore, in case of parameterization by marking, the unmarked PIM must be kept separately from markings. The unmarked PIM and markings can be combined into a marked model for the purposes of transformation if necessary.

Design Structure Matrix. Table 7 provides an overview of the dependencies between each of the models considered in our analysis so far. The columns of this table correspond to the columns of tables 1 to 6. When the table is read row-wise, the '⊠' mark indicates that the model that names to the row is affected by the models that name each of the columns. When the table is read column-wise, the mark shows the models that may be affected directly as a result of a modification in the model that names the column.

Table 7. Dependencies between models: Design Structure Matrix

	Application PIM	Abstract platform	Application PSM	Concrete platform	Transf. arguments	Transf. specification
Application PIM	N/A	⊠	⊠ through transformation	independence is engineered		
Abstract platform		N/A		independence is engineered		
Application PSM	⊠ through transformation		N/A	⊠	⊠ through transformation	⊠
Concrete platform				N/A		
Transf. arguments	⊠		⊠ through transformation	⊠	N/A	⊠
Transf. specification		⊠		⊠		N/A

DSMs exhibit an interesting property for our analysis: if we consider that there is a time sequence associated with the position of the elements in the matrix, then all marks above the diagonal are considered feedback marks [11]. Feedback marks require iterations in the sequence of tasks executed. DSMs can be manipulated to eliminate or reduce feedback marks, e.g., by reordering the sequence of elements in the matrix. It is also possible to group elements of the matrix into clusters, a technique which allows us to consider the set of elements of a cluster as a single module.

In the following section, we manipulate the DSM represented in Table 7 to show how the dependencies between models affect the design process.

4 Dependencies between models and the design process

Preparation and execution phase concerns. Table 8 shows a reordered DSM. The models that result from the preparation activities, namely, concrete and abstract platforms and transformation specifications are placed in the first three positions of the matrix. These models are grouped into a cluster, which represents the preparation phase. A second cluster represents the execution phase, grouping application PIM, transformation arguments and application PSM.

Table 8. Clustering dependencies with respect to preparation and execution activities

	Concrete platform	Abstract platform	Transf. specification	Application PIM	Transf. arguments	Application PSM
Concrete platform	N/A					
Abstract platform	independence is engineered	N/A				
Transf. specification	☒	☒	N/A			
Application PIM	independence is engineered	☒		N/A		☒ through transformation
Transf. arguments	☒		☒	☒	N/A	☒ through transformation
Application PSM	☒		☒	☒ through transformation	☒ through transformation	N/A

The absence of feedback marks above the diagonal formed by the preparation and execution phase clusters in Table 8 shows that the preparation phase does not depend on the execution phase. This result is made possible by requirements imposed on the preparation phase. These requirements are described in the cells of tables 1 to 6 that correspond to the cells positioned above the diagonal formed by the two clusters. Failure to satisfy these requirements would imply the presence of feedback dependencies, which would require revisiting the preparation phase. The absence of feedback marks above the diagonal formed by the preparation and execution phase clusters can be summarized by the following design rule:

Changes in PIM, PSM or transformation arguments must be accommodated in PIM, PSM or transformation arguments, but not in the abstract platform, concrete platform nor transformation specification.

Table 8 also reveals the absence of feedback dependencies within the preparation phase, since, within the cluster, no feedback marks appear above the diagonal. The same, however, cannot be said of the execution phase: modifications in the application PSM may affect the PIM and transformation arguments. The presence of feedback dependencies in the execution phase is addressed through iteration in the execution phase. An iteration in the execution phase allows a designer to gain insight into the implications of design decisions at the PIM-level for the application PSM, which may result in adjusting the PIM in a subsequent iteration.

However, for the design process to advance towards a stable application PIM, it is necessary that the dependencies between PSM and PIM should eventually decrease. Eventually, the application PIM must be such that it does not depend on design decisions that constrain the choice of target platform. This constitutes an important requirement for the iterative approach in the execution phase.

Multiple levels of models. We continue our analysis by considering the dependencies between the models at three different levels related by transformation. Table 9 shows the dependencies between the various models. These dependencies are clustered for each pair of consecutive levels of models, i.e., a cluster for models of levels 1 and 2 and a cluster for models of levels 2 and 3. This DSM is build by reapplying the transformation pattern, which explains the isomorphic nature of the dependencies in the two clusters.

Table 9. Clustering dependencies with respect to levels of models

	Abstract platform Π_1	Application PIM M_1	Transf. specification T_1	Transf. arguments a_1	Abstract platform Π_2	Application PIM M_2	Transf. specification T_2	Transf. arguments a_2	Concrete platform Π_3	Application PSM M_3
Abstract platform Π_1	N/A									
Application PIM M_1	⊠	N/A				⊠				
Transf. specification T_1	⊠		N/A		⊠					
Transf. arguments a_1		⊠	⊠	N/A	⊠	⊠				
Abstract platform Π_2					N/A					
Application PIM M_2		⊠	⊠	⊠	⊠	N/A				⊠
Transf. specification T_2					⊠		N/A		⊠	
Transf. arguments a_2						⊠	⊠	N/A	⊠	⊠
Concrete platform Π_3									N/A	
Application PSM M_3						⊠	⊠	⊠	⊠	N/A

The table shows an overlap between the two clusters. This overlap indicates that the design activities in the different levels are not completely independent, and that the intermediate model PIM forms the ‘interface’ between the two clusters, as could be expected.

5 Classifications of models

This section concludes our analysis by classifying the various models and design decisions according to the following dimensions of separation of separation of concerns:

- platform-independent and platform-specific concerns;
- application-independent and application-specific concerns, which correspond to preparation and execution phases concerns, respectively; and,
- transformation-independent and transformation-specific concerns.

Figure 4 places the different models according to the first two dimensions. Three levels of models are depicted.

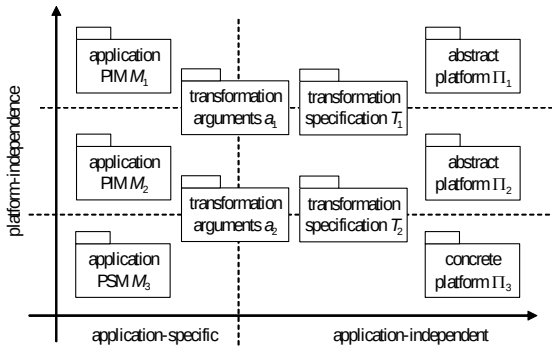

Fig. 4. Dimensions of separation of concerns and models

In Figure 4, transformation specifications are placed in the boundary between two levels of platform-independence. This is to denote that transformation specifications rely on the (abstract) platforms of both source and target levels of models (see Table 2 and Table 4). In addition, transformation specifications may also capture some transformation rules which are independent of the target platform.

Similarly to transformation specifications, transformation arguments are also placed in the boundary between two levels of platform-independence. In addition, transformation arguments are placed in the boundary between the application-specific and application-independent concerns area. This is to denote that arguments may be application-specific (see Table 1), but may also capture application-independent design decisions. Application-specific transformation parameterization is used to improve the generality of transformation specifications with respect to specific applications. Application-independent transformation parameterization is used to improve flexibility of transformation specifications in general, e.g., to cope with to variation in user requirements that are not captured in the source models but that are to be addressed during transformation. An example of an application-independent transformation argument determines that, irrespective of the application model, all application parts should be allocated to the same unit of deployment of the target platform.

In addition to the dimensions considered in Figure 4, we can also classify models related in a transformation step as *transformation-independent* or *transformation-specific*. This classification is relative to a transformation specification. In a transformation step, the source application model is transformation-independent (with respect to a transformation specification from that level of models), since it relies on an abstract platform, which is itself transformation-independent (see Table 6). The target application model and the transformation arguments can be classified as transformation-specific. This can serve as a guideline to determine whether design decisions should be captured at the source application model level or at either transformation arguments or the target application model level.

6 Main conclusions and directives

From the analysis of the relations between the various models, we can conclude that:

- *Feedback dependencies between execution and preparation phases can be avoided by addressing generality requirements at the preparation phase*. Failure to address these requirements results in cycles between the execution and preparation phases;
- *Platform-independent and platform-specific models are interrelated, their dependencies defined by transformation*. The interrelation between PIMs and PSMs is addressed through iteration in the execution phase. An iteration in the execution phase allows a designer to gain insight into the implications of certain design decisions at the PIM-level.

Our analysis leads to the following directives for the design process:

- *Changes in PIM, PSM or transformation arguments must be accommodated in PIM, PSM or transformation arguments, but not in the abstract platform, concrete platform nor transformation specification.*
- *Dependencies between PIM and PSM are handled by iterations in the execution phase, leading to a stable application PIM that does not depend on platform-specific design decisions.*
- *Interdependent design decisions must be captured at the same level of platform-independence. Since some design decisions are platform-specific, this imposes constraints on the organization of models at different levels of platform-independence. We have illustrated the consequences of interdependent design decisions with an example in [1].*
- *The classification of models according to the various dimensions of concerns serves as a guideline to determine in which models design decisions should be captured.*

References

1. Almeida, J.P.A., Dijkman, R., van Sinderen, M., Ferreira Pires, L.: On the Notion of Abstract Platform in MDA Development. In: Proceedings Eighth IEEE International Conference on Enterprise Distributed Object Computing (EDOC 2004). IEEE CS Press (2004) 253–263
2. 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 Seventh IEEE International Conference on Enterprise Distributed Object Computing (EDOC 2003). IEEE CS Press (2003) 112–134
3. Baldwin, C.Y, Clark, K.B.: Design Rules, Volume 1, The Power of Modularity. MIT Press, Cambridge, MA (2000)
4. Baldwin, C.Y, Clark, K.B.: Modularity in the Design of Complex Engineering Systems, Harvard Business School Working Paper Series, No. 04-055 (2004)
5. Gardner, T., Griffin, C., Koehler, J., Hauser, R.: A review of OMG MOF 2.0: Query / Views / Transformations Submissions and Recommendations towards the final Standard, ad/03-08-02, OMG (2002)
6. 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, Enschede, The Netherlands (2004) 43–51
7. Object Management Group: MDA-Guide, V1.0.1, omg/03-06-01, OMG (2003)
8. Object Management Group: UML 2.0 Superstructure, ptc/03-08-02, OMG (2003)
9. Steward, D.V.: The Design Structure System: A Method for Managing the Design of Complex Systems. In: IEEE Transactions on Engineering Management, Vol. 28 (1981) 71–74
10. Warfield, J.N.: Binary Matrices in System Modeling. In: IEEE Transactions on Systems, Man, and Cybernetics, Vol. 3 (1973) 441–449
11. Yassine, A., Braha, D.: Complex Concurrent Engineering and the Design Structure Matrix Method. In: Concurrent Engineering, Vol. 11, No. 3, SAGE Publications (2003) 165–176