Methods for the Development of Adaptive and Dependable Information Systems

Diana Carolina Gomez Hernandez

gomez@rhrk.uni-kl.de

Abstract. This paper presents methods that are suitable for the development and selection of information systems (IS) that comply with the exigencies and complexity of enterprise business processes. The challenge now, is to reduce costs by using methods that allow the adaptivity and dependability of an IS in a changing environment. The methods presented here are the Model Driven Architecture (MDA), developed by the OMG group, and the Enterprise knowledge Development Change Management Method (EDK-CMM). Both methods are used in important industry sectors, and they obey to their exigencies of having a structure in which an architectural separation of concerns in necessary. To accomplish this, they separate the logic of business from the information system specifications, by using modeling as an approach to support the development process.

1 Introduction

During the last years several methods for software development have been proposed, but many of them lack in some important aspects that an information system needs, such as maintenance, integrity, and low cost of development [16]. The use of an effective approach that complies with these requirements and the complexity of information systems nowadays became an undeniably necessity. The methods explained in this document show possible ways to fulfill these demands, as well as new requirements like dependability and adaptivity.

The main objective of a dependable information system is to satisfy user requirements, and deliver the specified application services during its operation time in a changing system context and environment [2]. An adaptive information system must be able to accommodate a diverse set of users [22], and changing business rules within an enterprise [17]. The Model Driven Architecture (MDA) enables the development of dependable and adaptive systems. The promise of MDA is to create application code from models, ensuring that the requirements specified during the first phases of the software development process are met by the final product. Hence, the errors produced when manually creating code from models are removed and consequently the final system fulfills a great part of the requirements. By using models with the definition of layers within a system, MDA guarantees that changes in the system

specification adapt quickly in the development process [1]. The Enterprise Knowledge Development Change Management Method (EKD-CMM) satisfies dependability and adaptivity in its process. It implements business processing reengineering, in which business models can adapt to the changing situations and decisions in an enterprise. Since the main goal of the EKD-CMM is to specify the requirements of an enterprise information system by using modeling, it provides integrity during its complete process [17], [21].

In this document we will present in detail both methods, by explaining their potential in the development process.

2 History of Modeling

Formerly, applications were developed by directly using machine code or assembler language in which the programmer had to use very complex instructions to create application functionality. A huge step forward was the introduction of the first programming language FORTRAN. With FORTRAN, programmers had the opportunity not only to write applications in a friendly way, but also to use a compiler to generate machine code [1]. Thereafter, the door for other high-level programming languages like C was opened. Although, with the use of these programming languages the problem of having a compiler to produce machine code was solved, a more structured way to developing code was still needed. Hence, programming languages with object-oriented structure like Java or C++ came up, allowing the programmer to easier structure the application into concepts or functionalities.

With the broad use of programming languages, the idea of modeling emerged allowing the software developers to observe a system application from a higher level of abstraction. With modeling, the developer first specifies the requirements of an application then structures and analyzes the logic of it and finally creates implementation code. Modeling has evolved because of two factors: the complexity and wide range of concerns that involve today's software systems, and to achieve higher levels of abstraction to allow humans to understand and develop these systems [6].

Modeling techniques are evolving according to changing business requirements [6], [17]. Nowadays, companies are improving their way for developing information systems in order to cope with maintenance, integrity and scalability of solutions. Modeling helps in different ways to accomplish this. Since information systems are an integral part of a business processes in an enterprise [17], they need to be developed in a structured way in which the technology used for implementation is completely separated from the specification of the logic of the business. With modeling it is possible to describe the interaction among different entities in an enterprise, for example, operating systems, business processes, actors and flow of information without considering implementation aspects. Additionally, one of the challenges ahead for an enterprise is to offer products and services that are suitable for the final client. In order to fulfill

this, modeling provides the possibility to validate software requirements according to the client demands.

Modeling also has evolved according to the need of software developers to have certain level of abstraction, in which the main characteristics of a certain problem are easy to identify and understand [6]. By only using the initial techniques of software development (assembler, high level languages, OO languages), it was difficult to identify the requirements and behavior of a system, and developers focused too much on the details of one specific implementation technology. Modeling languages like UML were created and have gained wide acceptance among software developers. Despite the use of UML is a powerful approach to design systems, a problem still remains: the result of modeling stays only in a document or may be in the developer's head (the translation from models to code still has to be done manually) [6]. Hence, the more the code evolves, the bigger will be the gap among the design and the implementation. Therefore, the challenge ahead for modeling techniques is to find ways to keep a direct connection between the code and the models. This is where methods like the Model Driven Architecture come in.

3 MDA Model Driven Architecture

The *Model Driven Architecture* (MDA) is a method used for developing software that separates the business logic from the platform in which the system is developed, maintaining the integrity of the system application during the entire software life cycle. MDA promotes the use of models in software development, by following the methodology of the Model Driven Development (MDD), in which the modeling of software is the base for the generation of system applications [4]. The OMG group defines MDA as *an approach to using models in software development in order to integrate "what you have built" with "what you are building" with "what you are going to build"* [1]. Hence, MDA guarantees that the models created in the software development cycle are used during each of the four phases of the development process: Analysis, Design, Implementation and Testing, by organizing them into an architectural framework of different levels and transformations [6].

3.1 MDA Objectives

The main objective of MDA is to separate the description and specifications of an information system from the platform in which that system is finally implemented. To achieve this, MDA separates the code layer from the modeling layer.

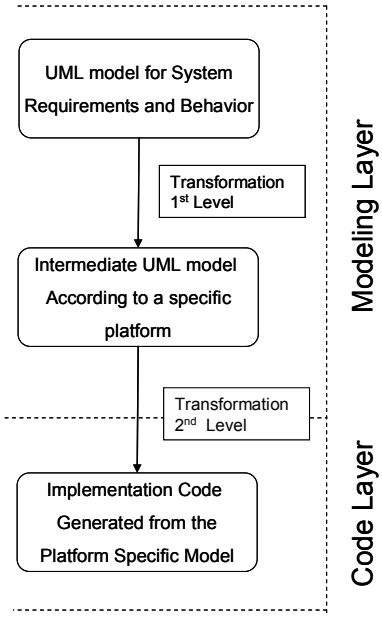

Fig. 1. Separation among MDA different layers

Figure 1 shows two types of transformations within two different levels [10]. In the first level a UML model for system requirements and behavior is transformed into a specific UML profile model, suitable for a specific platform. Although this model is specific for a platform it still belongs to the modeling layer. In the second level, the platform specific model is transformed into implementation code, making the connection between the modeling layer and the code layer. With this MDA is not only compliant to the requirements of the information systems nowadays [6], but also to the promise of having one day implementation code completely derived from models.

Another important objective of MDA is to provide a standardized way to develop and deploy applications. It offers tools and procedures implemented using known standards as XML and CORBA. Thereby, MDA improves the reusability, interoperability and maintenance of software applications [1].

3.2 MDA Models

For MDA, Models are the main instrument in the development process. They are not only used for documenting the process or guiding the implementation, but also to obtain programming code used for the implementation of a system. The OMG group defines five types of models used in the MDA:

Computation Independent Model (CIM)

The *Computation Independent Model* describes the requirements of the system including the environment in which it will be used. The CIM represents what the system is expected to do and is useful not only as an aid to understand a problem, but also as a source of a shared vocabulary to use in other models [1]. The CIM does not consider the internal structure or processing of a system [7], but it serves as an interface between the domain experts and the system architectures designers. An example of CIM is the UML use cases descriptions, in which the user's system requirements are explained.

Platform Independent Model (PIM)

A model can be platform independent regarding the two different layers in the MDA architecture. In the modeling layer, the *Platform Independent Model* specifies the structure and behavior of a system independently of the platform in which it will be implemented [1]. Additionally, the PIM is useful when validating the correctness of a system [3]. At this layer, the PIM is represented by several UML diagrams such as use case diagram, class diagram, sequence diagram, and collaboration diagram. In the code layer, a PIM specifies the implementation code model, independent of the CPU in which it will be running.

Platform Model (PM)

The *Platform Model* describes a platform by defining its services and technical concepts, and specifies the use of a platform by a system application [1]. The concepts described by a *Platform Model* are necessary when transforming a PIM into a PSM.

Platform Specific Model (PSM)

The *Platform Specific Model* is the result of a PIM transformation. It refines the PIM by using the specifications given by the PM, so to say the details of how a system uses a specific kind of platform [1]. Hence, a PSM can be defined as a PIM that includes all the details of a specific platform [6].

3.3 MDA Modeling Standards

There are several languages and standards that MDA uses to specify the models and transformations among them. The OMG group defines the following standards, to use with the MDA Procedures: The Unified Modeling Language (UML), the Meta-Object Facility (MOF), XML Meta Data Interchange (XMI) and the Common Warehouse Meta-Model (CWM). To better understand the standards, it is important to first introduce the concept of "Metamodel".

Metamodel

In order to provide a standard to specify the models that are used during the MDA development process, it is necessary to use *Metamodels*. A metamodel defines the rules and patterns needed to construct another model [8]. To better understand the concept of metamodel lets consider the UML Metamodel hierarchy in Figure 2.

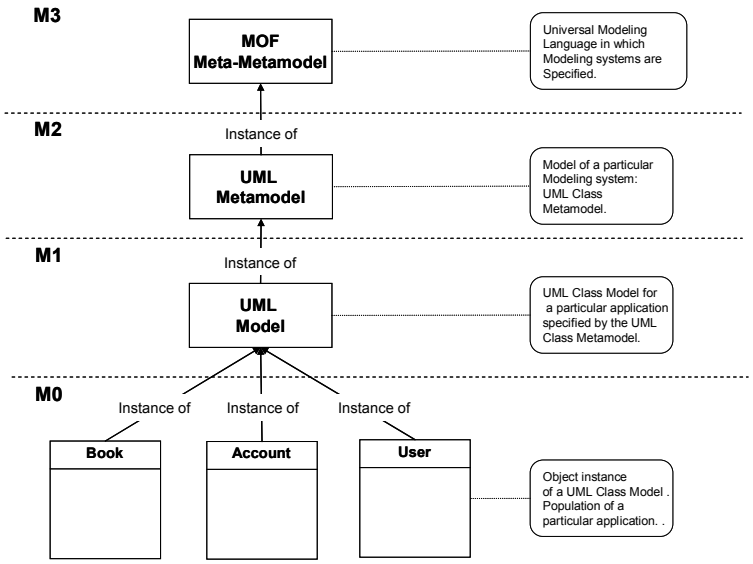

Fig. 2. Specification of the UML metamodel hierarchy

In Figure 2, we can distinguish 4 levels: M3, M2, M1, and M0. In the level M3 the metamodel language MOF is specified, as a base for creating UML metamodels. In the level M2 the definitions of an UML metamodel for a specific modeling system[1] are specified. At this level the UML concepts can be defined, such as class, association, and property. In the level M1 UML models specific for certain applications are specified by using the UML metamodel. At this level an UML class model contains the specification for classes such as Book, Account and User. In the level M0 the objects or instances of the classes defined by the UML class model are encountered [19].

Unified Modeling Language (UML)

Unified Modeling Language (UML) is a modeling standard proposed by the OMG group, and it is defined as *"a specification defining a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems"* [9]. UML is used in the MDA transformation process for many specifications

[1] Modeling System refers to a particular kind of modeling environment offered by UML, such as class diagram, sequence diagram, or activity diagram.

that will be defined in the level of models: Platform Specific Models (PSM) and Platform Independent Models (PIM) [8]. At the level of platform independent models, UML is used to specify the behavior of a system without including specific details of a platform. At the level of platform specific models an *UML profile* would be used. The UML profile allows including in the model the particular semantics of a specific platform by using stereotypes and tagged values, some of the UML extensions. [3], [13].

Meta Object Facility (MOF)

The Meta Object Facility (MOF) is defined by the OMG Group as *"an extensible model driven integration framework for defining, manipulating and integrating metadata and data in a platform independent manner"* [9]. By looking at this definition, we can infer that MOF is an extensible model integration framework, because it integrates every step of the software development process by standardizing the transformations from PIMs to PSMs [10]. MOF is used for defining, and manipulating metadata[2] and data in a platform independent manner. It is the language used to specify metamodels [5], [6], [19]. By expressing its models in MOF-based format, MDA allows [10]:

- Interoperability among applications: Models can be imported and exported from one application to another.
- Reusability of models defined during the process: The models are managed and stored in a repository together with relevant information used during the development process.
- Portability applied to models: Models can be translated into different formats, for example XML, to be transported across the network.
- Integrity during the development process: Models can be transformed and then used to produce application code.

XML Metadata Interchange (XMI)

XML Metadata Interchange is a framework defined by the OMG group with the purpose of *"defining, interchanging, manipulating and integrating XML data and objects"* [9]. XMI uses XML standards (XML Document, DTD and Schema) to integrate information among repositories of models, applications, data warehouses, and tools. It defines the format to interchange information between MOF format models and UML models, and also to mapping models from MOF to XML [8].

[2] Metadata is a term for data that describes certain kind of information. In modeling metadata specifies the rules to construct a metamodel [5].

Common Warehouse Meta-model (CWM)

The Common Warehouse Metamodel (CWM) is the OMG Group metadata standard that allows the interchange of warehouse metadata[3] [5], [9]. While metadata is used to maintain and manage the data warehouse, each part of the warehouse system such as warehouse tool, warehouse platform, and warehouse metadata repository, uses different metadata representation [5]. The CWM helps to integrate these different representations of metadata. It is based on the three standards explained above: UML, MOF and XMI. UML is used to design and obtain a graphical representation of metamodels and models, MOF is used to define metamodels using CORBA architecture, and XMI is the standard used to interchange information among warehouse metadata [5].

3.4 MDA Platforms

A platform can be defined as a framework[4] that provides certain functionality to support the development of software applications [1]. MDA defines two types of platforms: Vendor Specific Platform and Technology Specific Platform.

Vendor Specific Platform

In a vendor specific platform software applications are developed and launched within a set of rules and **specifications** particular for this platform. Examples of vendor specific platforms are IBM's Websphere for Java 2 Enterprise Edition (J2EE), or Microsoft's .NET. These platforms provide a specific functionality, but the system requirements and usage specifications differ from one to the other [1], [4].

Technology Specific Platform

A technology specific platform defines the methods and languages for implementing software, but the final implementation needs a vendor specific implementation for this technology. Examples of a technology specific platform are the standard CORBA, and Java 2 Enterprise Edition (J2EE) [1], [4].

3.5 MDA Transformation

A transformation in MDA is the process of converting a model into another model of the same system and it is applied to models within different layers. These layers are defined for the MDA specifications as the MDA *Viewpoints*. A viewpoint is defined

[3] Metadata in data warehouse can be classified into two different types: source metadata (repository specifications, source schemas) and data staging metadata (data transmission, transmission times and transmission results) [20].

[4] Framework is defined as an adaptable and expandable system of collaborating software units that provide an available functionality for a general set of tasks.

for the MDA guide as “a technique of abstraction that uses a selected set of architectural concepts and structured rules to focus on particular issues on a system” [1]. To better understand the concept of viewpoints, it is worth to make a comparison between the different viewpoints and the different phases of the software life cycle. Figure 3 shows how each viewpoint corresponds to a different phase of the software development:

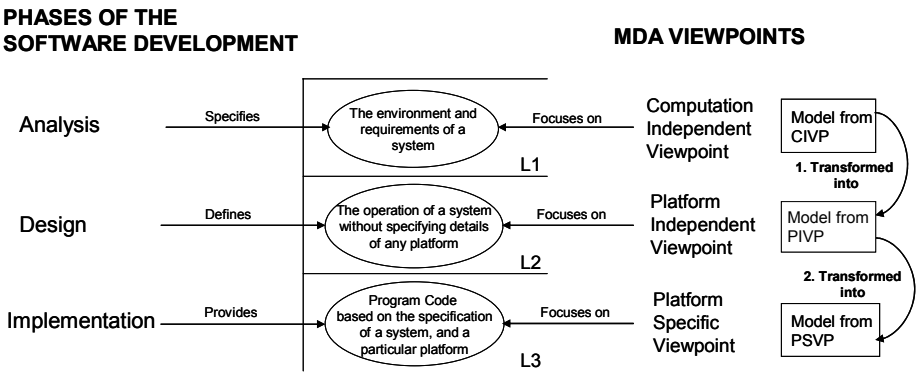

Fig. 3. Comparison between MDA Viewpoints and Software Development Phases

Thereby, models are transformed from one viewpoint to another, from a higher level of abstraction to a more specific level of abstraction [6], [13]. Based on these definitions it can be inferred that *transformation* is the process in which models *evolve* from one viewpoint to another. In Figure 4 the model in the higher level of abstraction is distinguished as a PIM and the model result in the lower level of abstraction could be a PIM or a PSM.

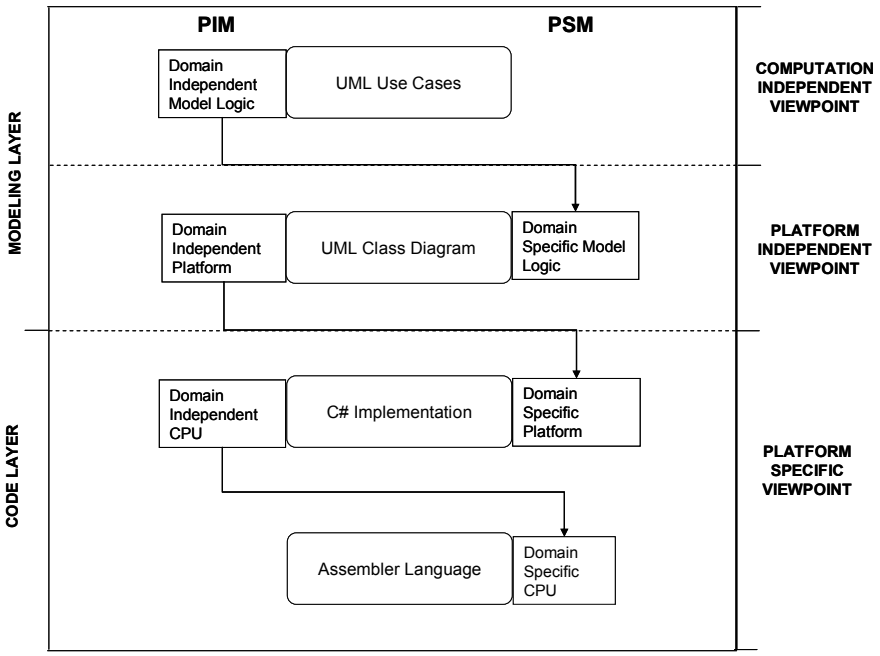

Fig. 4. Transformation among three Viewpoints.

Figure 4 shows how the models can be PIM or PSM according to different domains[5]. In the application logic domain, the UML class diagram is a PSM for the application logic domain, but a PIM regarding the code implementation domain. Through transformation, this model becomes a PSM in the code implementation domain, and a PIM regarding the CPU domain. When applying the final transformation (compilation), this model becomes a PSM for the CPU domain.

3.5.1 MDA Transformation Mappings

The transformation process from one model to another is guided by mappings [1]. A mapping gives rules and specifications to transform one model into another of a specific platform. The most common approach to map models is the *Model Type Mapping*. In this approach a PIM model is defined by using a certain language and then transformed into a PSM defined by a platform specific model language [1], [3].

[5] A domain summarizes concepts from a certain subject matter [16].

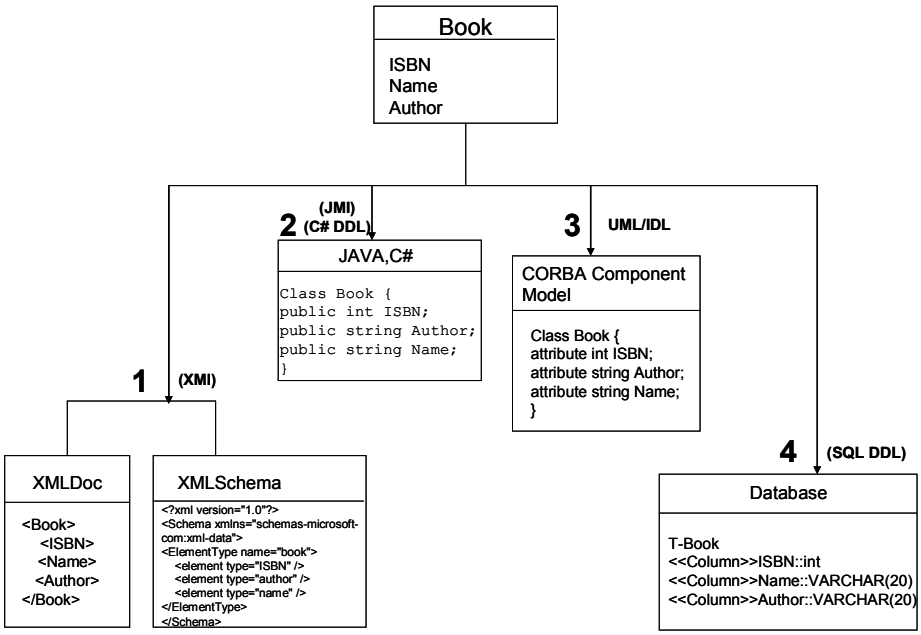

Fig. 5. Example of a PIM UML Model transformed into different PSM Models

Figure 5 illustrates four different kinds of mappings [6], [13]:

1. UML to XML
2. UML to Java
3. UML to CORBA Component Model CCM (Uses IDL Language).
4. UML to SQL Data Definition Language (SQL-DDL)

In each mapping a set of rules, algorithms and templates are defined by using descriptions of the language in which the final PSM must be implemented.

Another mapping approach is the *Model Instance Mapping* in which some elements of the PIM are identified to be transformed by choosing a specific platform. This approach uses *Marks* in a model, to indicate how PIM elements should be transformed into PSM elements [1]. A mark is useful for a transformation tool or performer to identify additional information that is needed to guide the transformation [13]. For example when there are a number of possible mappings for a model element, marks can specify which one of these mappings should be used. One example of a PIM marked is shown in Figure 6 [15]:

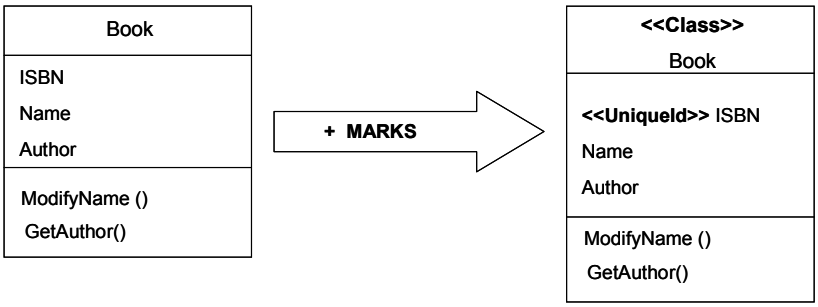

Fig. 6. Example of marks using a UML Book class

3.5.2 Outputs of a Transformation

The transformation process not only has a platform specific model as a result but also a *Record of Transformation*. A record of transformation stores, the specific parts of the mapping that were used to transform the elements, and the correspondence between the elements among the PIM and the PSM [1].

3.5.3 Examples of a PIM to PSM transformation

Using a UML Profile and Marks [1], [3], [15]

By using a UML profile with the marking technique, it is necessary to apply the following steps:

1. A PIM is defined
2. A Platform is chosen
3. The PIM is marked according to the specifications of the platform
4. Mappings are specified according to the platform. (A platform should offer specifications for making a mapping).
5. The transformation is performed: The marked PIM is mapped into a PSM according to the mappings.

In Figure 7 is shown an example of this case of transformation:

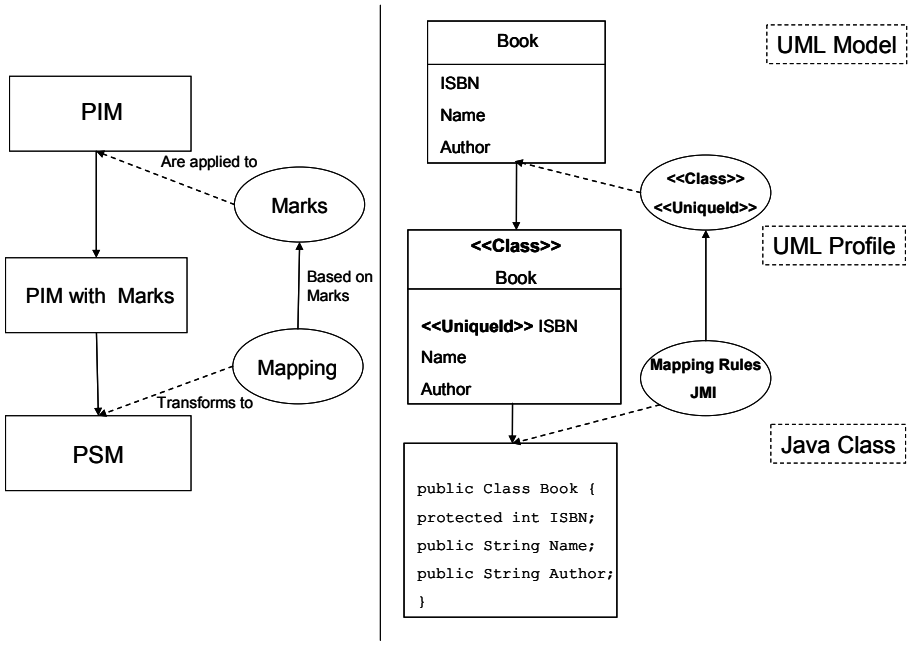

Fig. 7. Transformation from Book UML Class to Java Class using Marks

Meta-model Transformation [1]

The Meta-model transformation approach bases its methodology on the following steps:

1. A PIM is defined by using the concepts and the model elements defined by the chosen Platform Independent Meta-model.
2. A Platform is chosen, and the rules of the transformation are specified using a Platform Specific Meta-model.
3. A Mapping is defined according to the PIM Meta-model and PSM Meta-model
4. The transformation is performed: The PIM is mapped into a PSM using the mapping defined.

In figure 8 is illustrated an example of the steps explained above:

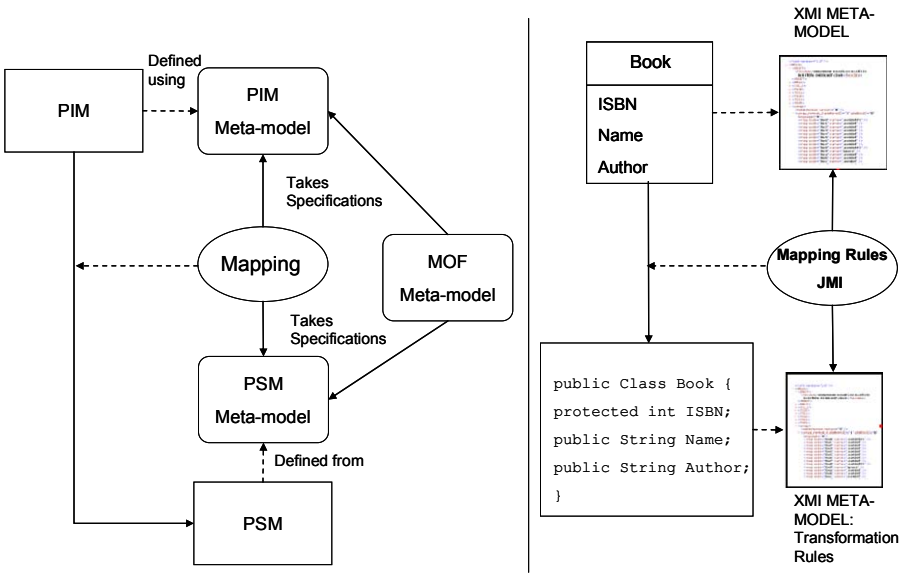

Fig. 8. Transformation from Book UML Class to Java Class using Meta-Model Transformation

4 A specific application of MDA: MDA in the avionics Industry [16]

We now discuss the use of MDA in the context of dependable and adaptive information systems by exploring an example of MDA in the avionics industry. This section will discuss: (i) the functionality blocks of the military aircraft industry, (ii) as well as their approach of using MDA including a short evaluation of how MDA improves quality in an environment that requires of dependable and adaptive systems.

Functionality Blocks of the aircraft industry

The military aircraft industry uses a MDA compliant environment in order to improve their existing method of software development, which was based on the "Waterfall method". This method follows the traditional steps of the software development: Analysis, design, code and testing. Since the military aircraft industry system has several blocks of functionality, MDA is a suitable method to integrate all these blocks in a standardized way. By using MDA, all system specifications are abstracted and then translated into several execution platforms. In order to do this, the notion of layers

defined in MDA is used to specify their different blocks of functionality as is shown in Figure 9.

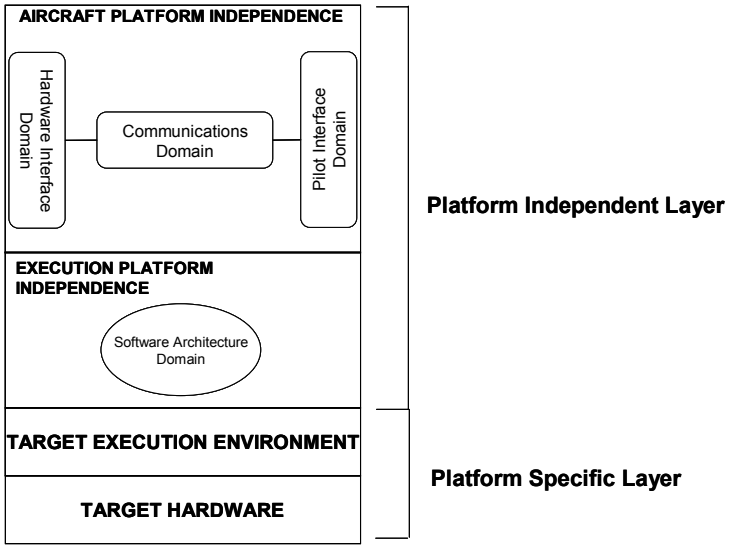

Fig. 9. Layers in the Avionics System

Figure 9 shows how the military aircraft system is structured. The platform independent layer is divided into two sub layers. One is specific for the aircraft system and the other is specific for the software architecture. In the aircraft specific layer are defined the hardware interface domain (which defines the applications of sensors and weapons), the pilot interface domain (the artifacts interface for communicating with the pilot), and the communications domain (the protocols for the aircraft's communication equipment). In the software architecture layer the mapping rules for transformation from PIMs to PSIs[6] (Platform Specific Implementations) are defined. The platform specific layers are the target execution environment layer and the target hardware layer. In the former operating systems, programming languages and processors are defined, and in the later the target hardware in which the system runs is defined.

In order to manage the creation and transformation of models, the aircraft system uses two different tools: the intelligent UML (iUML) tool and the Configurable Code Generator (iCCg) tool. The iUML tool is used to build models that are expressed in executable UML format (xUML). Hence, the tool allows the software developer not only to create models, but also to validate them (the xUML format can be transformed into executable code that is processed by a simulator to test a model). The iCCg tool is used to transform a model expressed in xUML format to implementation code. In order to do this, the iCCg tool needs mapping rules definitions.

[6] The concept of Platform Specific Implementation (PSI) is equivalent to that of Platform Specific Model (PSM) of the MDA.

The MDA Approach

The general process that the aircraft industry follows by using the methodology defined by MDA is exposed in Figure 10:

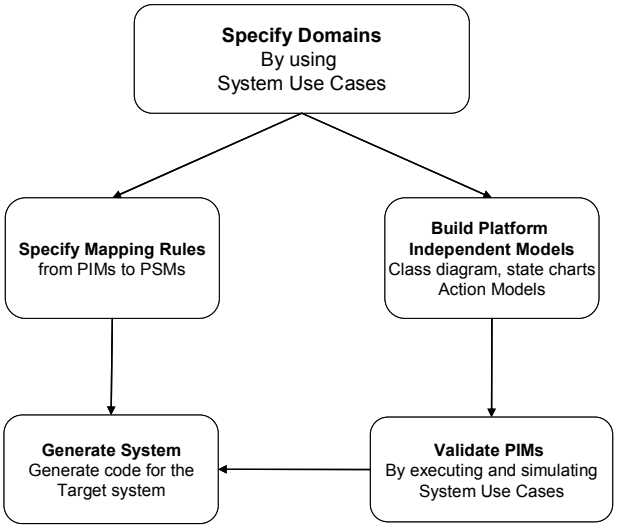

Fig. 10. MDA approach used in the aircraft system

Figure 10 describes the steps performed for model transformation. First, system use cases and domain use cases are defined to specify the behavior of the system. Second PIMs such as state charts, class diagram, and action models, are created to represent functionalities in each domain. In parallel, mapping rules for the transformation are specified. Third, PIMs are validated and tested by using the iUML tool. Fourth and finally, PSIs are generated by applying the mappings to the PIMs using the iCCg tool. In order to give an example of this process, a specific function of the military aircraft system is analyzed in detail: Transform a set of applications[7] into a set of processing modules.

[7] An application component in the avionics industry is defined as a set of processes that belong to certain domain and are active in an aircraft mission. Examples of processes are: take off, reconnaissance and attack. [16]

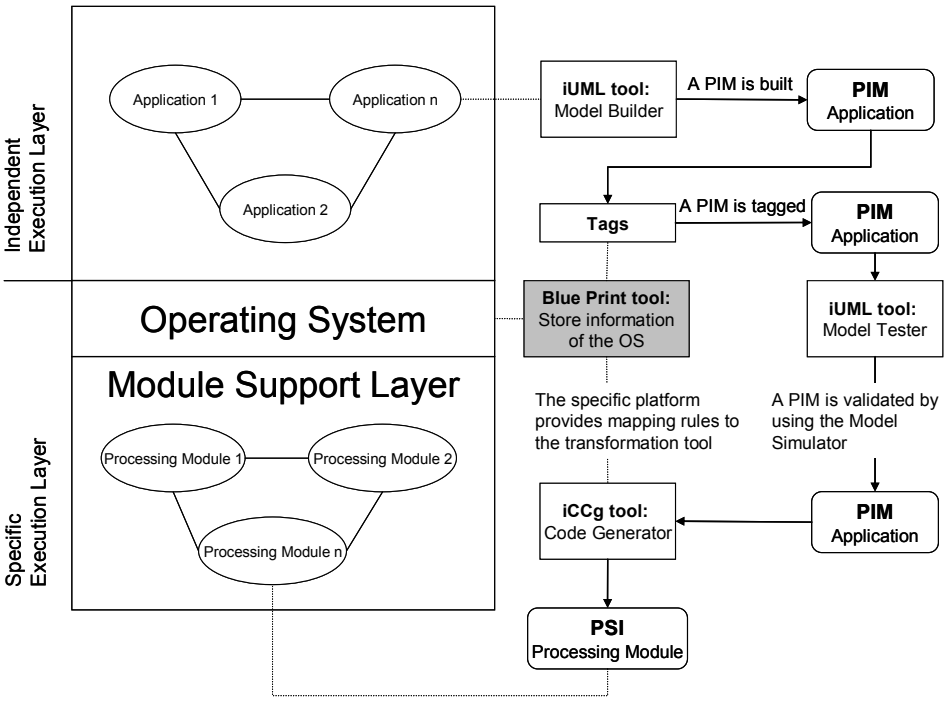

Fig. 11. Example of a transformation in the aircraft system

Figure 11 shows the process in which an application is transformed into a processing module. This is achieved by using the following strategy:

1 Represent the application component as a PIM by using the iUML tool.
2 The PIM is tagged to specify how the application is to be organized into modeling processes. (Here is used the marking technique from MDA).
3 The mapping rules are defined by using a runtime blueprint framework (which provides information of the OS) and incorporated in the iCCg tool.
4 The PSI model is generated by the iCCg tool to run in a specific platform.

MDA and quality

In their approach of using MDA, the military aircraft industry implements dependability in two ways: (i) by verifying the models in different domains and mapping rules used to transform these models, (ii) by performing an automated transformation process.

The models defined in the different platform independent domains and the mapping rules are validated against a number of use cases to show that the system requirements are fulfilled. Since both models and mappings are defined in xUML, it is possible to

perform a simulation test by using the iUML tool, in which models can show in advance the expected behavior of the system without considering the target specific platform (they are launched in an UML executable environment).

To achieve high quality systems, the military aircraft industry complies with different industry standards such as Generic Aircraft-Store Interface Framework (GASIF) defined by the SAE AIR 5532[8]. With the use of an automated transformation, the developers do not need to be aware of these technical specifications. To accomplish this, rules and specifications of the industry standards are included in the code generator tool, reducing the risk to introduce errors and inconsistencies when performing a transformation.

Since realiability and dependability are important attributes of the militrary aircraft systems, MDA methodology including the iUML and iCCg tools provides verification and validation within the development of the aircraft system applications.

5 The EKD-CMM Method

The Enterprise Knowledge Development Change Management Method (EKD-CMM) is a method used to choose the appropriate information system structure for an enterprise organization [17], [21]. In this method three different layers are distinguished, which are shown in figure 12:

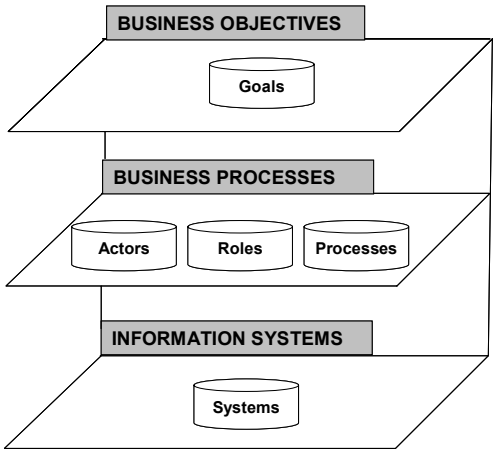

Fig. 12. Layers in the EKD method

Each of the three layers shown in figure 12 owns a set of models defined for a specific purpose. For example, the layer *business objectives* must contain a set of models that

[8] For more information please visit http:// www.sae.org

describe the goals of the enterprise (Enterprise Goal Model); while the business processes layer must contain a set of models that describe the processes in an enterprise, including actors and roles in each process (Enterprise Business Model) [21]. The information system layer contains the result of applying this method: The Enterprise System Model (ESM). The ESM contains the list of requirements of a suitable information system for an enterprise. In other words, it describes how to support business processes at operational levels [17]. In order to achieve this, this method defines several approaches by using a road map[9].

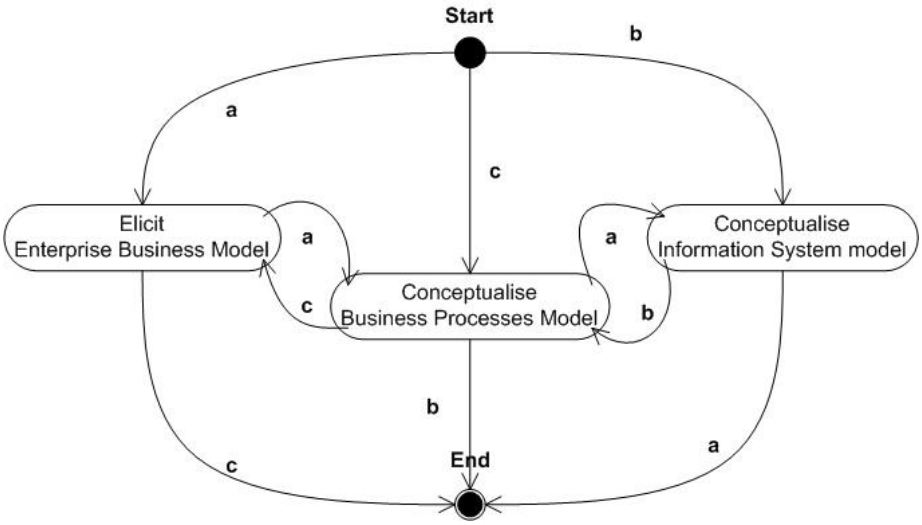

Fig. 13. Road Map defined by the EKD-CMM

Figure 13 shows how three ways in which the modeling process can be done in the EDK-CMM. The first one (a) Forward Engineering, describes an approach that goes from the layer of business objectives to business process and finally to the information system layer. The second one (b) Reverse Engineering, describes an approach in which the process model is described according to the requirements of an information system. The third one (c) Business Process Reengineering or Improvement goes from the business process layer to the business objectives layer in order to change specifications defined in one of both layers [17], [21].

In order to obtain the ESM it is required to start from a business process model. Therefore it is necessary to first follow the approach a), in which a business process model is specified from details given by the business objectives model. After having a business process model, Business Object Models (BOMs) must be defined. The BOM is the connection among the information system layer and the Business Process Layer. The BOM contains definitions of the information requirements only. To refine the

[9] A road map in the EDK-CMM context is an approach to describe different possibilities by going from a starting point to an ending point.

BOM with information of a specific information system, Business Process Rules (BPR) are defined at the information system layer. BPRs define the rules and operations that must be applied to a BOM in order to fulfill the information system requirements. The BOM after applying BPRs is called Technical Business Object Model (TBOM), which is the central model in the information system layer [17]. To create and define these models, EDK-CMM uses the ABC flow charter as a modeling tool. Since this tool does not support integration among models, an integrated computer tool suitable for enterprise modeling is currently being developed. Hence, the transformation process between BOMs and TBOMs is performed manually by now.

5.1 EKD-CMM compared to MDA

While EKD-CMM and MDA have different goals, they agree in the following aspects:

1. Both methods use the methodology of the model driven development (MDD), in which models are the base of the software development.
2. Both methods use transformations in order to obtain the model that is suitable for a specific platform or information system: In MDA a PIM is transformed into a PSM, and in EDK-CMM a BOM is refined to a TBOM , by using rules given by a specific platform, or information system.
3. Both methods involve and define the concept of layers as a base of their structure, but the context in which this concept is defined differs from one to the other.
4. Both methods promote the evolution of models as a promise to maintain the integrity in a system development environment.

Although MDA and EDK-CMM work in a similar way, they have different focuses. MDA is a method to develop software and EKD-CMM is a method that helps in choosing an information system suitable for an enterprise. In MDA, when applying transformations in models the final product is the implementation code or executable code (expressed by a platform specific model) that runs in a specific platform. In EKD-CMM the final product is an enterprise system model that considers the requirements of an information system but not its implementation.

6 Conclusions

The methods explained in this paper accomplish the demand of dependability and adaptivity of information systems, because they promote:

- Separation of Architectural Concerns: Since MDA and EDK-CMM use layers to define architecture of different levels, they separate the logic of business from the platform specification or implementation. To fulfill this, MDA defines three viewpoints. The computation independent viewpoint and the platform independent viewpoint contain models that specify the behavior and functionality of the system. On the other hand, the platform specific viewpoint contains models that

specify the functionality of a system for a specific software or hardware platform. In EDK–CMM three different layers are defined with the purpose of separating the business objectives and the business processes from the information systems.

- Integrity: The MDA and the EDK-CMM use transformations among models to unify the logic of business and the platform specification. That means, that the specifications of the behavior of a system defined in models or business processes are kept when they apply to a specific platform or information system.
- Validation: Both methods support the quality of their final products (PSM and ESM), by validating them against the requirements of the enterprise or the final user defined by models from the first layers of the development process.
- Maintenance: MDA and EDK-CMM support the quick adaptation to changes in the logic of business or the system behavior. Changes are implemented in the models that belong to the platform independent layers, reducing the costs to maintain an application.
- Modeling Standards: Modeling standards in MDA not only allow interoperability among models, but also in a future the possibility for models to adjust themselves dynamically according to changes of the system requirements. For example, when a new model is inserted into a model repository the other system components will adapt themselves to changes produced by that model being reflected directly in the system application [23].
- Modeling domains: By offering the possibility to create modeling domains, MDA and EDK-CMM are able to enable adaptive systems. With separating a system in different model domains, both methods allow to combine information from different domains in order to make a system self-sufficient .As an example of that we can consider the system hippie [22]. The system hippie is an adaptive system that is used to model user's preferences in a physical space i.e. a museum exposition. The preferences of the user are modeled according to different modeling domains, such as user's knowledge, interests, and movements. The user knowledge model helps to reduce redundancies of information that already has been shown to a user, the interest model learns information of the user's interests by using the key words of each section visited, and the movement model stores the track in which the user moves in the space. With all of this combined information, it is possible for the system to adapt the user's preferences according to a specific section in a specific area of interest.

Although the advantages of using MDA and EDK-CMM are numerous, the existing gap between the logic of business and the information system is still a challenge ahead for those who promote the implementation of these methods.

References

1. Miller Joaquin, Mukerji Jisnhu.: MDA Guide Version 1.0.1. OMG Group (2003).
2. Peleska Jan.: Formal Methods And Development Of Dependable Systems. Technical Report 9601. UniForM Workbench, University of Bremen (1996)
3. Architecture Board MDA Drafting Team.: Model Driven Architecture A Technical Perspective. OMG Group (2001) 2001-02-04

4. Muñoz Javier, Pelechano Vicente.:MDA a Debate. Universidad Politécnica de Valencia (2004)
5. Chang Daniel T, Iyengar Sridhar.: *Common Warehouse Meta-model Specification.* OMG Group (2001)
6. Brown Alan.: An Introduction to Model Driven Architecture:Part I: MDA and today's systems. IBM,(2004) http://www-128.ibm.com/developerworks/rational/library/3100.html
7. Cernosek Gary.: Next-Generation Model-Driven development. IBM Software Group (2004) http://www-128.ibm.com/developerworks/rational/library/feb05/cernosek/index.html
8. OMG Group Web Page.: MDA Specifications. OMG Group (2005) http://www.omg.org/mda/specs.htm
9. OMG Group Web Page.: Catalog of OMG Modeling and Metadata Specifications. OMG Group (2005) http://www.omg.org/technology/documents/modeling_spec_catalog.htm
10. OMG MOF Web Page.: OMG's Meta Object Facility. OMG Group, 2005. http://www.omg.org/mof/
11. SUN Microsystems Web page.: Java 2 Platform Enterprise Edition. http://java.sun.com/j2ee/index.jsp
12. Microsoft Corporation Web Page.: Microsoft.NET. http://www.microsoft.com/net/default.mspx
13. Brown Alan, Conallen Jim.: An Introduction to Model Driven Architecture:Part II: Lessons from the design and use of an MDA toolkit. IBM (2005) http://www-128.ibm.com/developerworks/rational/library/apr05/brown/index.html
14 Amor Mercedes, Fuentes Lidia, Valecillo Antonio.: Bridging the Gap between Agent-Oriented Design and Implementation Using MDA. Agent Oriented Software Engineering Proceedings 04-1 (AOSE 04-1). Springer Berlin 2004
15 Cepa Vasian, Mezini Mira.: Language Support for Mod**el**-Driven Software Development. Journal Science of Computer Programming (Elsevier). University of Darmstadt (2004).
16. Raistrick Chris, Bloomfield Tony.: Model Driven Architecture: An industry perspective. Architecting Dependable Systems II. LNCS 3069. Springer-Verlag Berlin Heidelberg (2004).
17. Barrios Judith, Nurcan Semil.: Model Driven Architectures for enterprise Information Systems. LNCS 3084. Springer-Verlag Berlin Heidelberg 2004
18. Metamodel Web Page . http://www.metamodel.com/
19. Colomb Robert, Chang Daniel, Boger Marko, Hart Lewis, Kendall Elisa.: Ontology Definition Metamodel. DSTC, Gentleware, IBM, Sandpiper Software (2004).
20. Kimball Ralph.: Meta Meta Data Data: Making a List of Data about Metadata and Exploring Information Cataloging Tools. DBMS (1998) http://www.dbmsmag.com/9803d05.html
21. EKD foundations web page http://crinfo.univ-paris1.fr/EKD-CMMRoadMap/foundationsEKD.html
22. Specht Marcus, Oppermann Reinhard.: Using modeling and adaptivity in nomadic information systems. Proceedings of the 7.GI-Workshop : "Adaptivität und Benutzermodellierung in Interaktiven Softwaresysteme." (1999).
23. Poole John D.: Model Driven Architecture: Vision, Standards and Emerging Technologies. Workshop on Metamodeling and Adaptive Object Models. ECOOP (2001)