CHAPTER 4

Model-driven Architecture (MDA)

The Object Management Group (OMG) has defined its own comprehensive proposal for applying MDE practices to systems development. This goes under the name of MDA (Model-Driven Architecture) [55]. We take MDA as a good exemplary MDE framework for two main reasons: first, MDA is a perfect case for explaining the MDE concepts introduced so far, as all the standard phases of a software development process such as analysis, design, and implementation are appropriately supported; second, given the importance of the OMG in the software industry, MDA is currently the most known modeling framework in industry. Without the aim of being complete, we report here some of the main contributions of the approach, which have become common knowledge and may ease the understanding of MDE in general.

MDA itself is not an OMG specification but rather an approach to system development which is enabled by existing OMG specifications, such as the Unified Modeling Language™ (UML®), the Meta Object Facility (MOF™), the UML Profiles SysML, SoaML, MARTE, and the CORBA® Component Model (CCM), among others.

The four principles that underlie the OMG's view of MDE are the following:

- Models must be expressed in a well-defined notation to foster effective communication and understanding of system descriptions for enterprise-scale solutions.
- Systems specifications must be organized around a set of models and associated transformations implementing mappings and relations between the models. All this enables a systematic design based on a multi-layered and multi-perspective architectural framework.
- Models must be built in compliance with a set of metamodels, thus facilitating meaningful integration and transformation among models and automation through tools.
- This modeling framework should increase acceptance and broad adoption of the MDE approach and should foster competition among modeling tool vendors.

Based on these principles, OMG has defined the comprehensive MDA framework and a plethora of modeling languages integrated within it.

4.1 MDA DEFINITIONS AND ASSUMPTIONS

The entire MDA infrastructure is based on a few core definitions and assumptions. The main elements of interest for MDA are the following.

- System: The subject of any MDA specification. It can be a program, a single computer system, some combination of parts of different systems, or a federation of systems.
- Model: Any representation of the system and/or its environment.
- Architecture: The specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors.
- Platform: A set of subsystems and technologies that provide a coherent set of functionalities oriented toward the achievement of a specified goal.
- Viewpoint: A description of a system that focuses on one or more particular concerns.
- View: A model of a system seen under a specific viewpoint.
- Transformation: The conversion of a model into another model.

As you can see, these official definitions are perfectly in line with the ones given in the introductory chapters on MDE and actually they represent the concrete incarnation and interpretation given by OMG to the various aspects of system definition. Based on these definitions, OMG proposes a whole modeling approach spanning methods, languages, transformations, and modeling levels. This chapter does not aim at providing a comprehensive coverage of the MDA framework. Instead, it is more aimed at giving an overview to the approach so as to let the reader understand the possibilities and potential usages.

4.2 THE MODELING LEVELS: CIM, PIM, PSM

As we introduced in Chapter 1, the level of abstraction of models can vary depending on the objectives of the models themselves. In MDA, three specific levels of abstraction have been defined, as shown in Figure 4.1.

- Computation-Independent Model (CIM): The most abstract modeling level, which represents the context, requirements, and purpose of the solution without any binding to computational implications. It presents exactly what the solution is expected to do, but hides all IT-related specifications, to remain independent of if and how a system will be (or currently is) implemented. The CIM is often referred to as a business model or domain model because it uses a vocabulary that is familiar to the subject matter experts (SMEs). In principle, parts of the CIM may not even map to a software-based implementation.

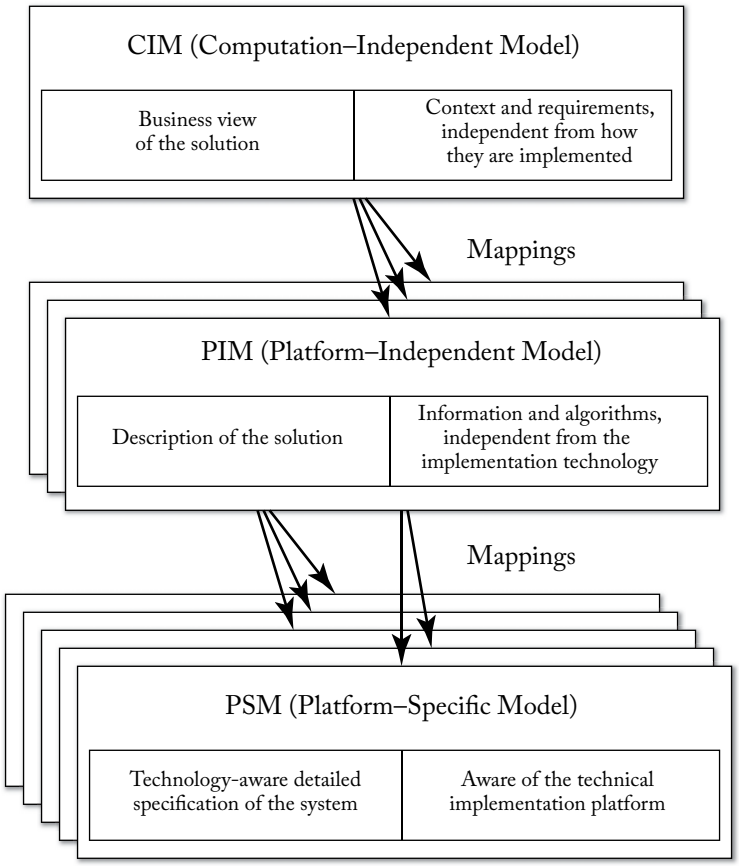

Figure 4.1: The three levels of modeling abstraction codified in MDA: computation-independent (CIM), platform-independent (PIM), and platform-specific (PSM) models.

- Platform-Independent Model (PIM): The level that describes the behavior and structure of the application, regardless of the implementation platform. Notice that the PIM is only for the part of the CIM that will be solved using a software-based solution and that refines it in terms of requirements for a software system. The PIM exhibits a sufficient degree of independence so as to enable its mapping to one or more concrete implementation platforms.

- Platform-Specific Model (PSM): Even if it is not executed itself, this model must contain all required information regarding the behavior and structure of an application on a specific platform that developers may use to implement the executable code.

A set of mappings between each level and the subsequent one can be defined through model transformations. Typically, every CIM can map to different PIMs, which in turn can map to different PSMs.

A concrete example of all three levels can be the following: a firm wants to implement an invoicing process. At the CIM level (Figure 4.2) the process is defined as a business process model listing the set of tasks to be performed and their dependencies. The PIM level will describe that the software application will do some parts of the job, in terms of information and behavior models (the access to a database and email sending could be fully automated, but shipping a signed paper invoice requires a clerk to mail the printed letter). Within this level, for instance, the details associated with a system implementation of the process must be detailed (e.g., Figure 4.3 shows a PIM description of the concept *Account*). Finally, at the PSM level, the implementation platform (whose description should be defined separately) for the process is selected and a set of precise descriptions of the technical details associated with that platform, including all the details regarding the invocations of the platform specific APIs and so on, will be provided within the models. For instance, Figure 4.4 shows the PSM description of the Account concept, including all the details of the specific implementation platform of choice (in this case, Enterprise Java Beans, EJB). The rationale is to keep at the PSM level all the details that are not relevant once the implementation moves to a different platform, thus keeping the PIM level general and agnostic.

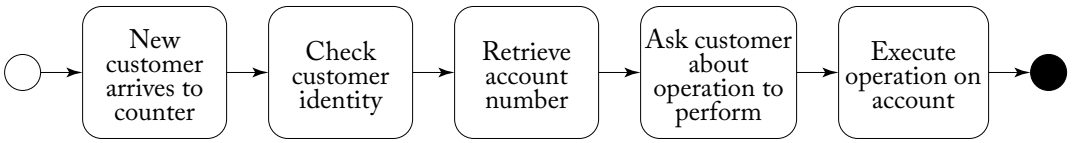

Figure 4.2: CIM example: the description of the business processes performed within an organization.

4.3 MAPPINGS

Generally speaking, a mapping consists of the definition of the correspondences between elements of two different models. Typically, a mapping is specified between two metamodels (i.e., modeling languages), in order to automate the transformations between the respective models. For instance, if the PIM level is described by a Petri Net, while the PSM is described by some UML dynamic diagram, the mapping must associate the concepts of Petri Nets with the corresponding ones in the UML diagram. Mappings between the metamodels allow us to build mappings between models, i.e., the traces (as described in Chapter 8). Mappings defined at the metamodel level are called *intensional*, while mappings defined at the model level are called *extensional* (because they are defined on the extension, i.e., instances, of the metamodel).

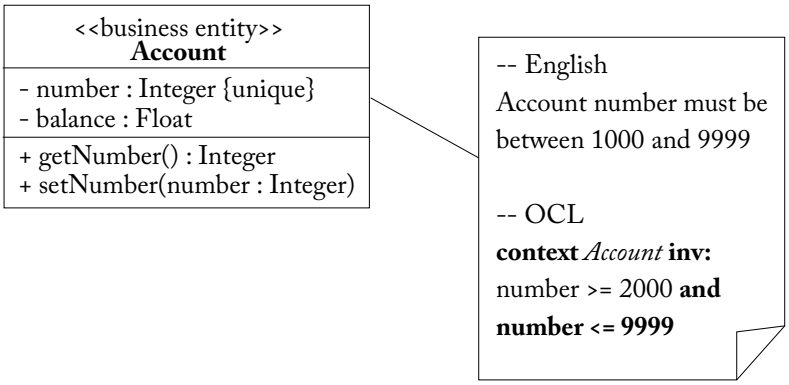

Figure 4.3: PIM example: the definition of a concept (*Account*), its properties, and constraints to be considered for any system implementation dealing with that concept.

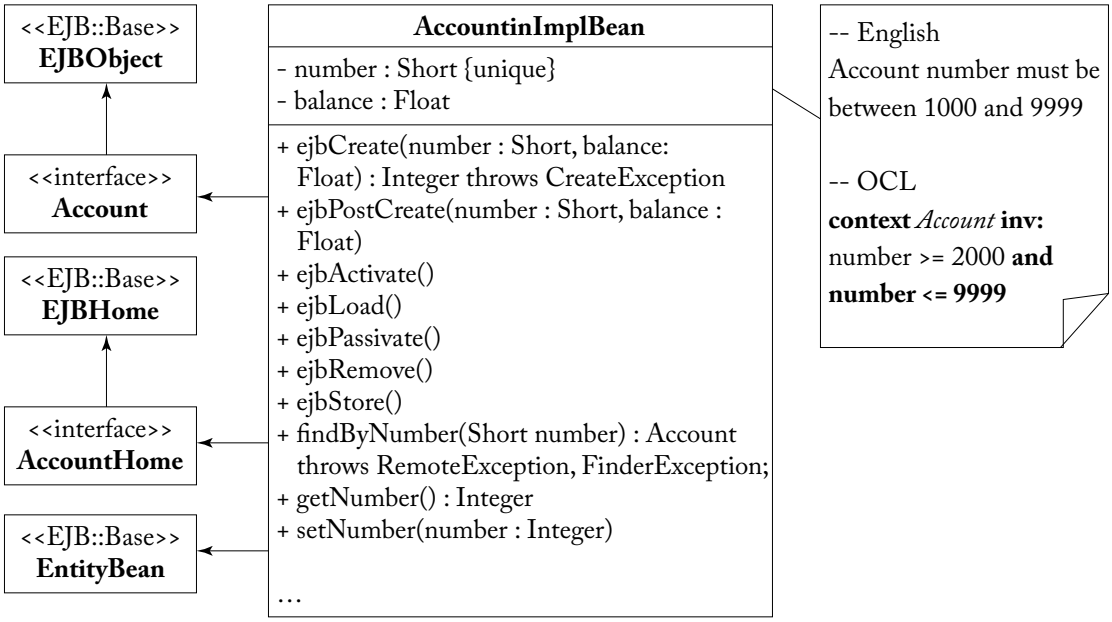

Figure 4.4: PSM example: EJB-based implementation model.

Mapping specifications can be defined in terms of: *weavings*, defining simple correspondences between metamodel elements; and *constraints*, describing the requirements for model transformations. From both specifications, model transformations may be derived.

The mapping gives rules and/or algorithms for the transformation of all instances of types in the metamodel of the PIM language into instances of types in the metamodel of the PSM language(s). Figure 4.5 illustrates this: the mapping from PIM to PSM is specified as a transformation defined between the platform-independent metamodel and the platform-dependent metamodel.

The mapping between different models (or modeling levels) can be implemented through transformations. Mappings can provide a conceptual specification for implementing the transformation between the different modeling levels. Obviously, the objective is to automate the mapping implementation as much as possible, thus avoiding expensive manual transformations.

In several cases, the mapping must take into account decisions that are taken along the development in terms of some details of the lower-level models. For instance, a mapping from PIM to PSM must take into account some decisions that apply to the PIM level based on choices at the PSM level. In these cases, a mapping defines annotations (*marks*, in MDA parlance) that will be used for guiding the transformations between the two levels. For instance, the transformation of a PIM to a PSM for the Enterprise Java Bean (EJB) platform will require a decision on which bean type should be created for the different PIM-level concepts. Basically, a mark represents a concept in the PSM which is applied to an element of the PIM in order to indicate how that element is to be transformed. A PIM plus all of the platform marks constitute the input to the transformation process resulting in a PSM.

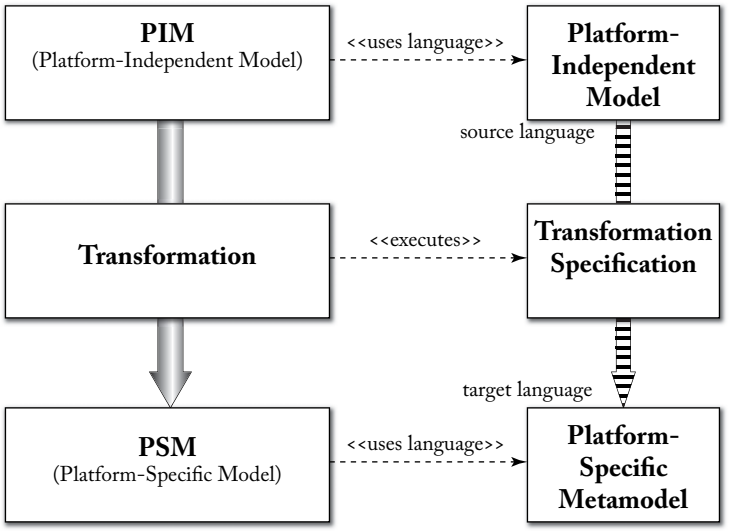

Figure 4.5: Representation of the transformation implementing a mapping between PIM and PSM levels.

4.4 GENERAL-PURPOSE AND DOMAIN-SPECIFIC LANGUAGES IN MDA

Besides the definition of the levels and mapping framework, the other main ingredients of MDA are the modeling languages used at each level. Indeed, one of the main OMG contributions is the standardization of a large number of modeling languages, spanning from general-purpose to domain-specific ones. Notice that the relationship between levels and modeling languages has cardinality M:N, in the sense that we can use more than one language to model the different aspects of a level, and vice versa a language can be used in more than one level.

In MDA, the core activity is the conceptualization phase in which analysis is conducted. First, requirements are codified as a PIM (or even CIM), and then the actual design of the application must be performed. This typically produces a PSM. In turn, the PSM is transformed into the running code, possibly requiring additional programming.

The cornerstone of the entire language infrastructure is the general-purpose language called UML (Unified Modeling Language), which is often perceived as a suite of languages, as it actually allows designers to specify their applications through a set of several different diagram types. Around and upon it, a set of domain-specific modeling languages (DSMLs) has been defined for addressing either vertical domains (e.g., finance, utilities, ecommerce) or horizontal domains (e.g., Web applications, mobile applications, service-oriented architectures), by exploiting either metamodeling techniques or the extensibility features provided within the UML language itself.

OMG supports UML Extensions through UML Profiles (as we will discuss in Chapter 6) and DSMLs by MOF. Typically, if you are dealing with object-, component-, or processes-oriented systems, you may reuse UML due to the already existing modeling support. If this is not sufficient, you can either use the inherent language extension mechanism of UML for a lightweight extension, or build a heavy-weight extension if the profile mechanism is not enough to tailor UML in the way you want, or build a completely new DSML from scratch. You may even use both, UML and DSMLs, on one level for different aspects or also for the same aspect (having a metamodel and a profile allows us to switch between UML and DSMLs) or on different levels. However, one should notice that adopting a design approach based on DSMLs is inherently different from using an approach based on UML (or any general-purpose language suite), as DSMLs will be definitely more tailored to the specific needs.

Notice that MDA is neutral with respect to this decision, in the sense that MDA is not favoring UML or DSMLs (although in the beginning of MDA, there was a strong focus on UML). In conclusion, MDA is about using CIM, PIM, and PSM: each of them can be defined either in plain UML, with a customized UML version, or using a DSML. Which languages you are employing, and if these languages are based on UML or not, is a different story.

4.5 ARCHITECTURE-DRIVEN MODERNIZATION (ADM)

Before closing the chapter on MDA, we also wish to mention another important initiative undertaken by OMG, that helps software developers in tackling the "reverse" problem, the problem of system evolution and modernization. In the same way that MDA is the OMG view of MDD, a corresponding initiative labeled ADM (Architecture-Driven Modernization) is addressing the problem of system reverse engineering and includes several standards that help on this matter. In particular, the Architecture-Driven Modernization task force of OMG [54] aims to create specifications and promotes industry consensus on modernization of existing applications, defined as any production-enabled software, regardless of the platform it runs on, language it is written in, or length of time it has been in production. The ADM taskforce aims at improving the process of understanding and evolving existing software applications that have entered the maintenance and evolution mode, so as to grant: software improvement, modifications, interoperability, refactoring and restructuring, reuse, migration, service-oriented integration, and so on.

The final aim of the initiative is to provide conceptual tools for reducing the maintenance effort and cost, and for extending the lifecycle of existing applications by enabling viable revitalization of existing applications through a model-based approach.

The ADM taskforce has defined a set of metamodels that allow describing various aspects of the modernization problem.

- The Knowledge Discovery Metamodel (KDM): An intermediate representation for existing software systems that defines common metadata required for deep semantic integration of lifecycle management tools. KDM uses OMG's MOF to define an XMI interchange format between tools that work with existing software as well as a general-purpose, abstract interface for modernization tools. KDM supports knowledge discovery in software engineering artifacts.

- The Software Measurement Metamodel (SMM): A meta-model for representing measurement information related to software, its operation, and its design. The specification is an extensible meta-model for exchanging software-related measurement information concerning existing software assets (designs, implementations, or operations).

- The Abstract Syntax Tree Metamodel (ASTM): A complementary modeling specification with respect to KDM. While KDM establishes a specification for abstract semantic graph models, ASTM establishes a specification for abstract syntax tree models. Thus, ASTM supports a direct mapping of all code-level software language statements into low-level software models. This mapping is intended to provide a framework for invertible model representation of code written in any software language (achieved by supplementation of the ASTM with concrete syntax specifications).

This set of metamodels is a valid instrument for enabling the typical model-driven reverse engineering (MDRE) and modernization scenarios also presented in Chapter 3 and reported in

Figure 4.6. Starting from the PSM of the system as-is, designers extract the PIM models through knowledge discovery, possibly exploiting 1-to-1 mappings to implementation language models as offered by ASTM. Then they work for making these models compliant with the requested compliance levels (e.g., described according to SMM) with respect to functional and non-functional requirements (possibly given at CIM level). Finally, they transform the system and process into seamless MDA-compliant solutions for the design and implementation of the to-be version.

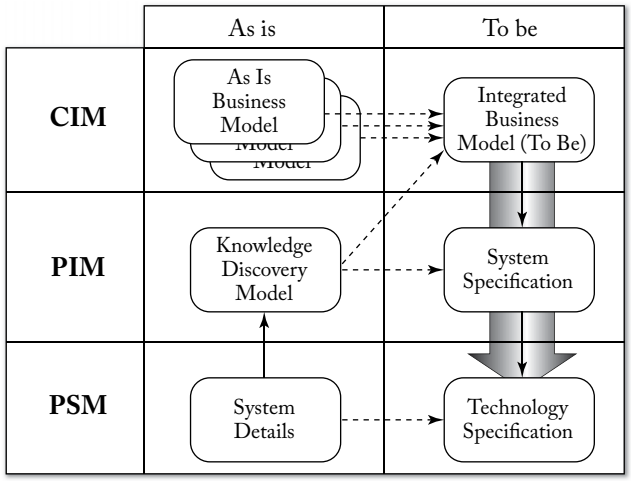

Figure 4.6: Overview of a typical MDRE scenario, as supported by ADM tools.