

PIM Definition and Description

Daniel Exertier, Benoit Langlois

Xavier Le Roux

THALES RESEARCH AND TECHNOLOGY

THALES Air-Traffic Management

MIRROR Pilot Programme

MIRROR Pilot Programme

Domaine de Corbeville
F-91404 Orsay Cedex France

19, rue de la fontaine
F-92221 Bagneux Cedex France

{daniel.exertier, benoit.langlois}@thalesgroup.com

xavier.leroux@thalesatm.com

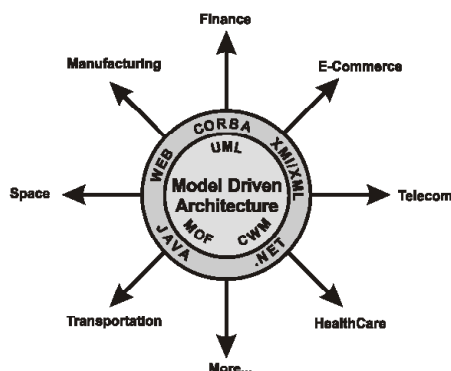
Abstract

Answering the requirement of the Model-Driven Architecture approach, as well as the general needs of engineering process definition, requires to precisely define the content and structure of the different models to be produced at project development time. For each model, the following questions must have an answer: What are its content and its objectives? What are the modelling concepts to be used to build the model? What are the well-formedness rules? How should the model be structured? How do I proceed to build the model?

Using the OMG's Model-Driven Architecture approach for software engineering, the article introduces Model Driven Engineering basic principles, and then, based on these principles, defines and describes a number of different platform-independent models (PIMs) for a software system, corresponding to different abstraction levels in the system modelling of a particular domain. The PIMs are fully defined from the well-known standard general purpose modelling language, i.e. the Unified Modelling Language (UML).

1. Introduction

To meet the enterprise needs, the Object Management Group (OMG) is standardising the Model Driven Architecture (MDA), introduced in the MDA "technical vision" document [11] as follows:



"The MDA defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. To this end, the MDA defines an architecture for models that provides a set of guidelines for structuring specifications expressed as models.

The MDA approach and the standards that support it allow the same model specifying system functionality to be realized on multiple platforms through auxiliary mapping standards, or through point mappings to specific platforms, and allows different applications to be integrated by explicitly relating their models, enabling integration and interoperability and supporting system evolution as platform technologies come and go."

The main objective of this article is to present a **Model Driven Engineering (MDE)** approach that puts the MDA vision at work. Through the development of modelling layers, architects and designers concentrate on business functionality and behaviour, down to the implementation layer where platforms (i.e. middleware...) strengths and differences come to the fore.

2. MDE approach

The MDA main keywords are : Models, Platform Independent, Platform Specific, Refinement, Mapping.

In the proposed approach, MDE complies with the basic **separation of concerns** principle, usually separating functional concerns and technical concerns within the development work organisation, in a classic Y process. It is brought one step further, separating technical concerns into platform-independent and platform-dependent technical concerns. It thus specifically integrates the separation of the following concerns:

- Functional aspects of the system vs. non-functional aspects. The functional aspects cover the domain aspects that are expressed, for example for the requirements, in terms of functions associated to services, business objects or business events. The non-functional aspect covers the qualities that are required on the system, without any domain functional considerations. The non-functional aspect covers for example qualities of service.
- Platform-independent vs. platform-specific aspects of the system. MDE/MDA promotes the separation between domain and technological concerns, leading to the definition of platform-independent and platform-specific models in the engineering chain. The merging between the platform-independent model and the platform description produces the platform-specific model.

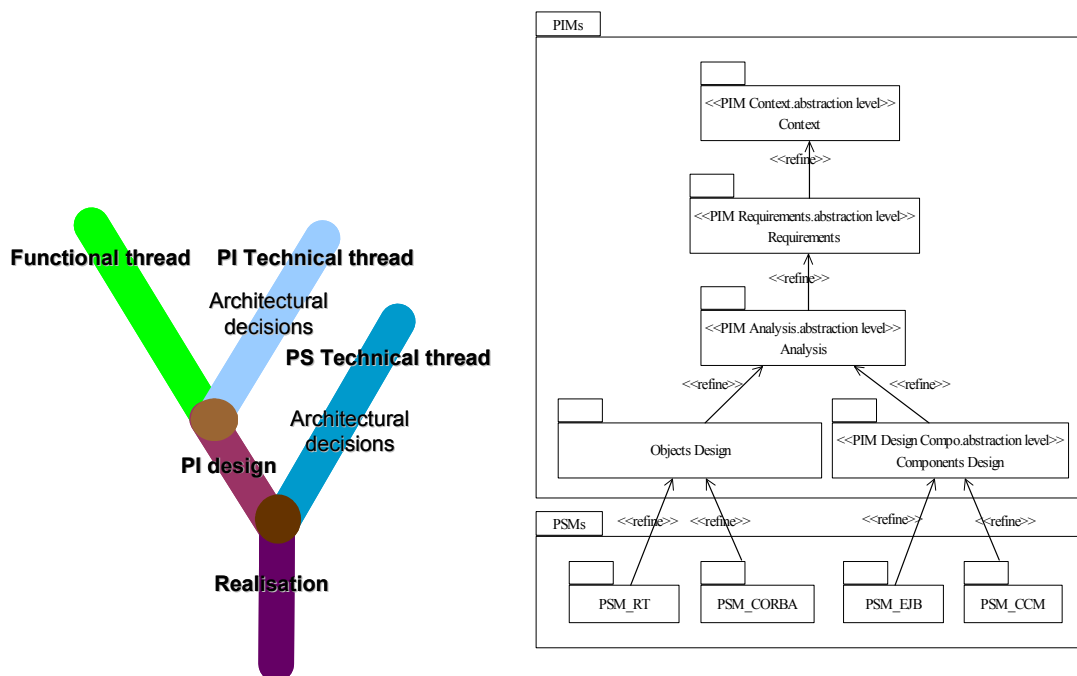


Figure 1: MDE Approach: 5 abstraction levels

This leads to the definition of a **chain of models** developed in a **refinement process**.

The MDE chain is intended to match both MDE principles (model refinement process and separation of concerns) and an organisation (model structuring, activities, work products, etc.) that enables to build a system from the requirements down to the code. This MDE chain can be adapted to meet specific organisation expectations.

The MDE/MDA process is organised around the production of a series of models:

- Upstream **Context and Requirements PIMs** provide a “black-box” viewpoint on the system under development; they describe the specification of the system frontier and detail the requirements on the system. The next models in the chain provide a “white-box” viewpoint of the system.
- The **Analysis and Components Design PIMs** provide platform-independent specifications of the system at two different levels of abstraction. The Analysis PIM models the functions of the system and the non-functional constraints that have been specified in the Requirements Model. The Component Design PIM models how the Analysis functions are to be implemented, in a

way that is platform-independent but not technology-free: this PIM relies on the usage of the component paradigm. Other concurrent Design PIMs may be defined, relying on different technological approaches (e.g. basic Objects Design).

- **PSMs** have been investigated with a main focus on the CCM platform. The CCM PSM refines the Components Design PIM. Additional studies have been carried out using EJB platform as the target platform, proving the PIM to PSM refinement concept.

In this approach, three considerations must always be kept in mind:

- The architecture is transversal and concerns every model. For example, the structure of the system is already defined in the Analysis PIM; this structure is adapted to the Component style in the Components Design PIM.
- Models are the primary elements produced during the development. All other development outputs (documents, code files, configuration files, etc.) are expected to be generated on the basis of these models, in an automatic or semi-automatic way. All elements to be modelled in a particular model are clearly defined using appropriate modelling rules.
- The models in the MDE chain are expected to be refined using tool assistance, based on model transformation rules. A transformation can add some technical know-how, especially in the lowest levels of abstraction. The model produced after tool transformation is then to be enriched with the level of information required.

3. UML specialisation

This section presents how the well-known standard general purpose Unified Modelling Language (UML) has been used as the foundation for developing PIMs, following the introduced MDE approach.

A solution for defining a modelling methodology dedicated to PIMs development is to **extend the UML syntax and semantics**. As a matter of fact, UML, being a general-purpose modelling language, does not propose any methodology guiding the development process.

The UML profile technique uses the extension mechanisms of Stereotype, Tagged Value definition and Constraint that can be attached to the UML model elements. For example, the MOF meta-class "Business Object" can be mapped in a UML Class bearing the "business object" stereotype. Then, each model element of a MOF metamodel can be mapped in UML with these extension mechanisms, and implemented in UML profile. This equivalence offers the possibility to build a metamodel at the MOF level and exploit it in UML with a UML tool, via UML profiles.

Consequently, this solution allows defining all the concepts (modelling rules) and the definition of how to handle these concepts (subprocess) that are needed at different levels of abstraction in a model-driven development methodology.

This approach takes advantage of the fact that the Unified Modelling Language (UML) standard is wide spread in the industry, that it is well documented in the literature, and that large development teams have already been trained to this modelling language. Furthermore, the majority of the industrial modelling tools are UML-based.

The UML specialisation provides the modelling principles and rules for the different models that are to be manipulated in a MDE/MDA software engineering process. It defines **Modelling Rules** as a combination of:

- A metamodel: it is a set of modelling concepts and relationships for describing the target domain of interest, including well-formedness rules. A metamodel defines abstract, logical modelling constructs; the concrete representation of these constructs (e.g., graphical representation in a diagram) is addressed separately. The metamodels are MOF-based – they are defined using the MOF metamodeling constructs (class, attribute, association, generalization, etc.). The metamodel defines which parts of the UML language is used, and specialised.
- A model structure : a set of pre-defined organisational packages (with the UML meaning) for storing the model elements at model building time, and for identifying the model framework within the project repository.

The software engineering process also defines, for the engineering of the models in the defined chain of models, one **Subprocess** per model. It identifies how the modelling rules concepts are handled along the development phase:

- Producing a model requires the collaboration of a set of distinct roles or logical responsibilities within the project team. A role represents a part a person can play in the production of the model. Each role performs activities corresponding to its specific responsibilities / competencies / concerns, which contribute to the production of the whole model.
- The engineering of a model follows general work decomposition. Each work definition comprises a set of activities, which are achieved by some project roles. For each of the activities, the subprocess identifies which elements of the modelling rules are consumed and produced.

Subprocesses are modelled along metamodels using OMG's SPEM, a the standard language for modelling processes (see [13]).

4. Platform Independent Models

Considering the UML specialisation approach, the different modelling rules and subprocesses of the above presented MDE approach are presented, as an example, in this section.

4.1. Context PIM

In the MDE approach, the **Context PIM** is supposed to be provided by the System Engineering activity but figures however in the MDE chain. The Context Model is the model presenting the uppermost view of the system to be developed. Its stake is to define clearly the scope of the system to be developed. The Context PIM: 1) introduces the system, 2) presents its goals (for what the system is clearly aimed to), 3) describes the business principles that structure the system (*e.g.* compliance to standards), 4) describes the external actors that interact with the system, 5) introduces the high level services required to interact with the system as key behaviours of the system, 6) and defines the business objects.

The Context clarifies the stakes of the system, its principles and its frontiers (mutual responsibilities) with the others systems; its interest is to make the system understandable by inexperienced people.

The Context PIM considers the system as a "black box". It is the input to elaborate the Requirements Model. The modelling of the Context PIM includes three major activities.

Define the system

The definition of the system constitutes the first step of the Context PIM construction. In this activity, The system to be developed is identified. The goals, the business principles and the high-level services or events offered by the system are described.

Define the system boundaries

A fundamental point in system and software engineering is to delimit the system boundaries, *i.e.* its responsibilities, and to clarify what is exchanged between the system and its surroundings. The following sub-activities are carried out in parallel:

- Identification of the actors. Each actor establishes relationships with the system through the solicitations of (high level) services on the system according to its own interests. Reciprocally, the system may solicit actors (high level) services.
- Identification of the business objects (as key concepts). Business objects clarify what the system uses from and produces for the actors, *i.e.* what they exchange for a set of clarified interests. Business objects are also these entities that the system is known to handle.
- Identification of data flows between the system and the actors.

Define the main system functionalities

The main functionalities of the system are also modelled. The technique used for this description can be compared to high-level use-cases, as coordinated exchanges between the system and the actors; in this analogy, every use-case represents a functional area.

Exchanges between the system and the actors, showing how the system behaves when stimulated are grouped according to functional areas. The internal behaviour of the system shall not be depicted here, only its behaviour, as observed from the outside. At the Context PIM level, only the main interactions between the system and the actors must be clarified. The interest is here is to roughly understand the behaviour of the system, taken as a whole.

4.2. Requirements PIM

The **Requirements PIM** specifies the external view of the system, using a "black box" viewpoint. In other words, it clearly describes how the system is **required** to behave when stimulated. The stake is twofold: 1) to build a model of client expectations with a clear vocabulary, 2) to have a unique requirement description that all subsequent models will be able to use. For example, an Analysis Model models the functionality specified in the Requirements model; a Platform-Specific Model can use the OS type – as defined in the Requirements Model – to produce a configuration file.

According to the separation of concerns, the Requirement Model contains two kinds of requirements:

- Functional requirements that are: the services, the business objects and events produced and consumed by the system and the actors interacting with the system, and the functions that are described using capabilities (*i.e.* functional "use cases").
- Non-functional requirements that are: qualities of service (*e.g.* response time, fault tolerance), qualities of development (maintainability, reusability, flexibility, demonstrability) and platforms specific requirements (*e.g.* OS).

Functional and non-functional requirements are described independently in order to address fully each time a specific concern. They are related at a second stage: a relationship specifies the conditions (a kind of contract) to be satisfied to associate a functional and a non-functional requirement: for example, a first functionality F1 has a response time (non-functional requirements) of 1 second (a condition to be respected) and a second functionality F2 of 0.5 second (another condition to be respected).

Corresponding textual requirements may be handled in the UML modelling tool, or using an external requirement management tool (such as DOORS). In that case, textual requirements must be traced to the appropriate elements of the Requirement PIM (functional areas, business objects, etc.). From this upper level of abstraction, by using traceability links, it becomes possible to establish a tooled impact analysis and to elaborate specific traceability matrix towards the subsequent models implementing the system at lower levels of abstraction.

The modelling of the Requirements PIM includes three major activities.

Specify the functional requirements

- Enrichment (refining) of the Context PIM system and actors services and events, as well as business objects. The exhaustive list of services an events is identified and interrelationships between business objects are modelled, expressing contractual information.
- Identification and organisation of the capabilities. A capability is a "functional use-case" (the non-functional considerations are addressed in another dedicated activity). They are organised according to the functional areas identified in the Context PIM. Capabilities help to discover, enrich and validate the model elements that represent the functional requirements of the system (business objects, business events, services). They are described with behaviours, modelled by activity graphs, and scenarios, modelled by sequence diagrams, compliant with the modelled behaviour of the system.

Specify the non-functional requirements

Identification and organisation of the forces. A force expresses a high-level non-functional requirement. A force may express a QoS (quality of service) but also a quality of development (*e.g.* usability for the system administrator, maintainability) or a platform constraint (*e.g.* OS type, maximum number of threads, memory size). QoS's are expressed by the definition of QoS Characteristics, using the OMG UML profile standard, or can use dedicated standard UML standard such as SPT (Scheduling Performance and Time) for Real-Time systems.

Relate the functional and non-functional requirements

When the concerned functional and non-functional requirements are modelled, they are inter-related in a specific context, so as to set their values. In a first step, the relationship between functional and non-functional elements is expressed at the force and capability level. In a second step, the QoS values are expressed, *i.e.* the QoS Characteristics are given a QoS Value for a given context, and applied to the functional elements of the model, where required.

4.3. Analysis PIM

The **Analysis PIM** specifies the internal view of the system using a "white box" viewpoint, without any technological nor software consideration, and still complying with the separation of concerns principle. It holds the functional specification of the system, focused on domain and application areas, and complies to the principle of separation of concerns (functional aspects of the system versus non-functional aspects). The functional aspect describes the internal entities of the system (with classes, attributes, packages, etc.), its functions (with operations) but also its boundary (external interfaces). The non-functional aspect is developed with the notion of QoS (Quality of Service) refining QoS requirements that are partitioned and allocated to the functional elements. Each Analysis model element conforms to the Requirements Model and justification information is expressed in the model by the definition and maintenance of traceability links between the elements of two levels of abstraction.

The Analysis Model is precisely described and is seen as the most durable model, being free of any platform or even "physical" architectural style consideration.

The modelling of the Analysis PIM includes three major activities.

Maintain the external interfaces

This activity consists in continuously analyse and maintain the external interfaces between the system and the actors, as they were specified in the Requirements PIM, for guaranteeing an overall consistency of the system integration in its over-system (*i.e.* the system that contains the system).

Perform the domain analysis

Here is built the core functional architecture of the system. This activity is well covered by state of the art literature and methodologies, with a number of well-proven practices being proposed in the public domain. The presented approach complies with those.

But the approach ensures that the Analysis PIM fulfils the capabilities expressed in the Requirements PIM. In a use-case approach, and for traceability and impact analysis concerns, functional realisations are created refining, one for one, the capabilities identified in the Requirements PIM. Every single functional realisation verifies, with compliant scenarios, that analysis classes realise correctly the capability.

Perform the QoS analysis

In parallel to the domain analysis, the non-functional aspect specified in the Requirements PIM with Forces and QoS are also analysed and refined in the Analysis PIM. The QoS characteristics are usually the same ones as those expressed in the Requirements PIM. The QoS values are refined and applied more precisely on the functional elements of the model.

The approach also ensures that the Analysis PIM fulfils the forces expressed in the Requirements PIM. Non-functional realisations are created refining the concerned force. Each non-functional realisation verifies that the analysis classes realise correctly the force.

4.4. Component Design PIM

The **Component Design PIM** represents a platform-independent solution expressed in terms of Software Components (component, interface, port, connector). This solution (the "how") meets the system's functional and non-functional requirements modelled in the Analysis Model (the "what").

This model, still being a PIM, specifies the common knowledge required to produce the next models specific to a Component platform, such as CCM or EJB.

The component design engineering process refines an Analysis PIM into a Component Design PIM. The functional and QoS requirements, as modelled in the Analysis PIM, and their relationships are refined at this level to form the model of a solution, for a distributed components type of platform. The modelling of the Distributed Components PIM includes four major activities.

Partition the system

The architecture of a software subsystem identifies a set of architectural elements, here components, which collaborate to achieve the system's functional and non-functional requirements. The objective of this activity is to specify this decomposition. The architect actually identifies the subsystem's components and their required and provided services.

Perform the component boundary design

As defined by UML2.0, a component is a modular, deployable and replaceable (pluggable) part of a system. It encapsulates its internal part and exposes a set of interfaces. It is composed of a declaration (its interfaces) and an implementation (its internal part), as well as a connecting part (connection between communicating components).

When the system has been partitioned and the interfaces established, the ports of a component can be defined. There are two kinds of ports: provided ports for accessing to the services of the component, required ports for negotiating with other components. A provided port is linked to an interface to offer the services of the interface.

Perform the component internal design

When the boundary of a component has been defined, its internal design can be performed: ports of a component (even the required and the provided) are linked to internal parts of the components (parts could be classes or subcomponents). Concerning all other design issues, classical design approaches are to be applied (with classes, associations, etc.).

In the same approach than in the analysis, for answering traceability and impact analysis concerns, functional realisations verify that the (Requirements) capabilities are correctly realised. A design realisation here addresses components, which complies with the Analysis PIM, where realisations involved the components underlying classes.

Perform the components logical deployment

Components collaborate to reach functional and non-functional requirements of the subsystem. However, some of the non-functional requirements can only be reached by component distribution over a network. The component logical deployment is defined to respond to this need. It can be seen as a specification of the distributed architecture to answer those requirements.

5. Conclusions and perspectives

The presented approach allowed to perform the MDE/MDA proof of concepts, applying the MDE/MDA technology on a real Air Traffic Management (ATM) case. On the way to have an industrial MDE/MDA environment for developing software systems and products, further work must be achieved, taking the technology developed as a foundation, and its application on the ATM Domain as its return on experiment.

Concerning PIMs and their experimentation, identified further issues are threefold:

MDA Core technology enhancement

- Traceability management must be enhanced, by precisely specifying the different kinds of traceability: intra-model traces (traceability with requirements and decisions), and inter-model traces (refinement).
- Languages must be defined and/or standardised (OMG's MOF2.0 Query/View/Transformation) for defining standard and interoperable models well-formedness rules, models transformation rules, etc.

MDE Coverage enhancement and extension

- Both Model Driven Software Engineering and Model Driven System Engineering are to be fully covered, including transition from System to Software Engineering. Other kinds of engineering (*i.e.* Agile, etc.) are also to be tackled.
- Quality of Service aspect was only partially tackled in a first iteration, only a subset of the OMG standard being used. The modelling approach to Quality of Service should be studied deeper, for reaching a better degree of ease of use and ease of exploitation.
- The architectures modelling itself should also be studied: meta-models dedicated to architectures definition must be defined, allowing to model different kinds of architecture models, at different levels of abstraction.
- The approach concentrated only on the development engineering phase, it did not cover the other phases such as testing.

MDE/MDA IDE definition and development

- In an MDE/MDA approach, engineering tools are required to drive, assist and control the development process. Going further than the tool prototypes developed so far, an industrial, supported, MDE/MDA tool chain must be specified and developed as in an Integrated Development Environment (IDE). The IDE tools should communicate through a standard tool exchange bus. The MDE/MDA IDE must take into account MDA methodology definition, UML modelling, requirements management, traceability management, model transformation, document and test generation, configuration management. The IDE should propose efficient multi-user and multi-site development environment, model repository management, multi-model projects management, model merging facilities, particularly for large software system developments.
- As part of the MDE/MDA IDE, a model transformation engine must be provided, allowing automatic or semi-automatic model refinement, pattern unfolding, as well as model browsing, filtering and viewing, dedicated to different model developer profiles.

References

- [1] S. Ambler. *Process Patterns: Building Large-Scale Systems Using Object Technology*, Cambridge University Press, October 1998, 582 pages.
- [2] A. Cockburn. *Writing Effective Use Cases*, Addison Wesley, October 2000, 304 pages.
- [3] D. D'Souza, A. Wills. *Objects, Components and Frameworks. The Catalysis Approach*, Addison Wesley, October 1998, 816 pages.
- [4] F. Dubois. Notation and semantics for deployment and configuration, Deployment and configuration support for distributed PNO applications. In EURESCOM Projekt P924, Deliverable 2, <http://www.eurescom.de/~pub-deliverables/P900-series/P924/D2/>, July 2001, 120 pages.
- [5] H. Eriksson, M. Penker. *Business Modeling with UML: Business Modeling at Work*, Wiley, February 2000, 480 pages.
- [6] P. Ferdinandi. *A Requirements Pattern: Succeeding in the Internet Economy*, Addison Wesley, November 2001, 528 pages.
- [7] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, January 1995, 395 pages.
- [8] I. Jacobson, G. Booch, J. Rumbaugh. *The Unified Software Development Process*, Addison Wesley, February 1999, 463 pages.
- [9] B. Meyer. *Object-Oriented Software Construction*, Pearson Higher Education, April 1998, 480 pages.
- [10] J. Odell. *Advanced Object-Oriented Analysis & Design Using UML*, Cambridge University Press, February 1998, 264 pages.
- [11] OMG. *Model Driven Architecture (MDA)*, Document number ormsc/2001-07-01, July 9, 2001, 31 pages.
- [12] OMG. *Meta Object Facility (MOF) 2.0 Core Proposal, Revised Submission to OMG RFP ad/01-11-14*, January 6, 2003, 117 pages.
- [13] OMG. *Software Process Engineering Metamodel Specification, Version 1.0, formal/02-11-14*, November 2002, 98 pages.
- [14] Rational Software Corporation. *Rational Unified Process, Version 2001A.04.00*, Copyright © 1987 - 2001.
- [15] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*, Addison Wesley, November 2002, 624 pages.