MDA-components:
Is there a Need and a Market?

Steve Mellor, Laurent Rioux, Jean Bézivin

OMG document ad/03-06-01:
OFTA/MDA users SIG/MDA working Group report
http://www.omg.org/docs/ad/03-06-01.pdf

Common meeting to:

the OMG MDA Working Group,
the OMG MDA User Group
the OFTA Software Model Engineering Group

Tuesday 3 June 2003, Paris

Organized by:

Steve Mellor, Project Technology Inc.
Laurent Rioux, Thales TRT
Jean Bézivin, ATLAS, (IRIN & INRIA), University of Nantes

Is there a need and a market for MDA components? This is the theme of the joint meeting organized by the OMG MDA working group, the OMG MDA user group and the OFTA Software Model Engineering Group.

The OFTA (Observatoire Français des Techniques Avancées, French Observatory for Advanced Techniques, http://www.ofta.net/) is a French non-profit organization that studies emerging technological fields that are likely to make an important impact, in the near future, on industrial practices; the Ecole Polytechnique Alumni Association founded it in 1982. The domain of studies is general industrial technologies, not only Computer Science. At the end of 2001, professor Marc Dupuis, head of OFTA, decided to launch a new study group on Software Model Engineering. This follows nearly thirty previous groups that have worked since 1982 on such different topics as molecular electronics, expert systems, nanotechnologies, neural networks, hybrid materials, fuzzy logic, molecular optoelectronics, biomimetism and materials, software components, supraconductivity applications, and much more. A group is created whenever a new technology seems in a position to create real changes on industrial practices. The group has a period of study of about two years and is supposed to provide industry and government bodies with a conclusion report at the end of this period. Each group, made on invitation only, is composed of about twenty experts from industry and academy and meets every six to seven weeks.

The subject of interest of this Software Model Engineering group is closely related to the MDA initiative launched at the end of 2000 by OMG. This is why we took the opportunity of the OMG technical meeting in Paris, to invite some French companies and organizations to present their work and position on this subject of MDA components. This fits well with the objectives of both the MDA working group and the MDA user group. Some of the experiences described in the following contributions were presented and discussed during the meeting.

The notion of an interoperable MDA component is progressively taking shape. There are many examples of such domain dependent or independent artifacts that a company could buy and adapt to its specific needs. At the center of this evolution, the notion of "transformations as assets" seems to start playing a key role. The panel on the subject of the emerging MDA component market was a good starting point that needs to be pursued. Laurent Rioux chaired this panel with the participation of Philippe Desfray, David Frankel and Jean-François Perrot.

Paris, June 2003

Steve Mellor
Laurent Rioux
Jean Bézivin

Contributions :

"MIRROR Project: The MDA experimentation and the deployment at THALES"
Serge Salicki, Thales, Corbeville

"Experimenting the MDA approach in France Telecom",
Mariano Belaunde, France Telecom, Lannion

"MDA experience and deployment at Sodifrance"
Erwan Breton, Yvan Gallison & Frédéric Madiot, Sodifrance, Nantes

"MDA experience and deployment at TNI/Valiosys"
Gilles Pitette, TNI Valiosys, Brest

"Using MDA to achieve seamless system to software processes"
Jean Philippe Lerat, Sodius, Nantes

"MDA experience University of Mulhouse: Model Driven Web Application Engineering"
Pierre-Alain Muller, University of Mulhouse

" MDA at LIP6"
Jean-François Perrot, Marie-Pierre Gervais & Xavier Blanc, University Paris 6, Paris

"MDA experiences and deployment at Tool Object",
Martial Christment, Tool Object, Toulouse

"The MDA vision at INRIA",
Jean Bézivin, Patrick Valduriez, Jean-Marc Jézéquel, Raphael Marvie & Jean-Marc Geib
INRIA Atlas, Jacquard & Triskell Groups, Nantes, Lille & Rennes

"MDA: Some Lessons Learned from Data Base Technology and Design"
Mokrane Bouzhegoub, PRISM, University of Versailles

"MDA for Distributed Real-Time Embedded Systems: experience and deployment at CEA"
François Terrier, Sébastien Gérard, CEA, Gif sur Yvette

Panel:

"MDA-components: is there a need and a market?"
Panelists : Laurent Rioux, David Frankell, Jean-François Perrot, Philippe Desfray

THALES

MIRROR Project:
"The MDA experimentation and the deployment at THALES"

Abstract

MIRROR is a THALES Pilot Program, addressing the Software Engineering aspects, and focusing on 2 main strategic issues expressed by the Business Units of THALES :

(1) Preserve the software investment by fighting technology obsolescence (technology change more quickly than domain and business functionalities),

(2) Capitalise on Domain, and services, but also on software development & maintenance practices engineering.

MIRROR aims at supporting THALES Business Units in the evolution of their methodologies towards Model Driven Engineering (MDE). MIRROR claims that MDE, putting the concept of model on the critical path of the software development, will turn the role of models from contemplative to productive by separating the technology and the domain and managing this separation throughout the development. A better control of the development chain (traceability, quality, consistency, impact analysis) and an automatisation of parts of development (weaving and merge between domain and technology aspects, code, documentation, test generation) will help the Business Units during long life cycle programs.

The technology used in MIRROR is essentially based on standards : UML for models notation, UML Profiles for models extensions, SPEM for process notation and MDA (Model Driven Architecture, the OMG initiative) for Platform dependent and platform independent aspects. This technology is necessary to control software development and enhance software quality and productivity. It includes:

- Core principles for defining system and software MDE development processes;
- MDE-aware methodologies, based on formalised concepts and processes and capitalising existing state of the art and institutionalised best practices, providing guidance to 'models developers' in using the general purpose UML modelling language;
- MDE-aware tooling, based on tool vendor independent technology, providing assistance to 'models developers' through the whole development lifecycle.

Although MIRROR MDE eases and clarifies system and software 'models developers' tasks, the underlying technology encompasses a large set of complex and evolving technologies.

The present document describes the emerging part of the technology, allowing the reader to reach a global understanding of MIRROR MDE and what it brings in practice.

MIRROR objectives

The Object Management Group (OMG) defined in [MDA 2001], the basic concepts, in fact the "technical vision", of Model Driven Architecture (MDA). This reference documents introduces MDA 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."

.

From Models …

The current systems require the expertise of many specialists to master a whole set of software technologies that are in perpetual evolution and not necessarily interoperable. On another hand, business domains evolve slowly compared to the technology. A promising solution is the adoption of a process where the automation of code generation from UML models takes a major role. This approach, focusing on models instead of wordy documentation and manual coding, is known as **Models Engineering**, which aims at unifying the software activities, from the requirements down to the code, around UML models management. Models Engineering offers the means of better mastering complexity, evolution and heterogeneity of underlying platforms (hardware and middleware).

The main stakes for Models Engineering are:

- *Productivity improvement by capitalisation on the domain*: development teams have only to focus on domain and not on low-level technical considerations (e.g. middleware, hardware concerns). Those activities can be concentrated on their domain expertise, like requirement expression, domain analysis, software design or system tests.
- *Productivity improvement by capitalisation on the platform*: The platform field (technology and hardware) is very large (OS specificities, interoperability by ORB or not, languages, specific approaches of programming and design like Component, software non-functional solutions like fault-tolerance…) and implementation solutions are numerous. With a short-term view, it is possible to capitalise around dedicated or in-house platform solutions, but with a long-term view, it becomes necessary to build software on a unified platform[1]. The solution is the adoption of existing standards, and the involvement in in-progress standards. The interest is to use the capitalisation of technological know-how, compliant to standards, realised by CASE or methodological teams, technology-vendors and tool-vendors.
- *Quality improvement*: Modelling, i.e. abstracting a problem, and associated tooling, i.e. model checking or automatic document and code generation, offer the opportunity to improve the software quality. The objective here is to ensure a higher quality of the developed software, in terms of understandability, maintainability or stability, using models. The stake is that quality becomes a condition for the software industrialisation.
- All these stakes really contribute to *ROI* (Return On Investment) improvement: The first form of capitalisation is on the domain in order to have a durable and maintainable

[1] By analogy, before UML there were many modelling notations (e.g. OMT, HOOD, SA-RT, MERISE). Here, a unified platform is a UML development platform unified by standards (e.g. UML profiles, CCM, Quality of Service, SPT for real-time, but also XMI for model inter-exchange or a common language of model transformation). The stake is the portability of the UML models on different UML tools.

application. The second form of capitalisation is technologic, with standards and tooling, in order to improve the productivity.

…to Model Driven Engineering (MDE)…

The OMG, a major actor of the standardisation (UML, CORBA…), has taken this opportunity into account and has radically oriented its strategy of platform federation on models engineering, promoting a **Model Driven Architecture (MDA)** approach. The MDA defines the major principles for an MDE process. What appears, as the most important is the PIM/PSM distinction. A PIM (Platform-Independent Model) represents a business domain model without any target platform consideration. A PSM (Platform-Specific Model) is another model dependent on a platform (EJB, CCM, .NET, etc.), resulting from the projection of a PIM on such platforms. The successive models constitute a chain of models dealing with the successive steps of development.

MDA proposes an infrastructure for the model engineering with a set of principles, e.g. the distinction PIM/PSM, and with a federation of standards, some of them are specific to MDA (such as model transformation language) and others are UML standards (e.g. UML profiles for quality of service, for real-time). Therefore, this infrastructure must be customised to a specific context with its constraints and its objectives and where model engineering is promoted. **Model Driven Engineering (MDE)** aims at customising MDA to meet these constraints and these objectives.

A Model Driven Engineering (MDE) approach consists in defining a (system, software, agile, etc.) engineering approach based on the concepts depicted by MDA. For example, developing a piece of software in a peculiar Model Driven Software Engineering consists in developing models, rather than code, and integrating them.

MDE will carry a lot of impacts and benefits on the development of software intensive systems :

- First of all, models will become the unique reference from which all the artefacts of the whole life cycle can be generated (documentations, test, …), and so will prevent from inconsistent references.
- The multiple views underlining the concept of model will drastically help the development of complex system by the simple principle of separation of concerns. Each expert can stay concentrated on his area of expertise (domain expert, architect expert, real time expert, implementation expert, …).
- Furthermore, and certainly the most important, is that MDE will provide, organize, and manage an explicit repository of models of the development know-how. Indeed, by promoting an explicit separation between technology dependant and technology independent models, MDE will gather all the stable know-how of the experts involved in the development. A corollary to this is that stable standards on know-how can emerge and be shared as well. Then MDE will provide an automatic way of weaving, assembling, transforming and refining them up to implementation models specific to the particular execution infrastructure.
- Among its other benefits, MDE will enable to specify interoperability between services or other applications through linked models and generate bridges connecting implementation on multi-segment platforms.

In short, MDE puts OMG's MDA at work

MIRROR initiative :

MIRROR is the THALES Pilot Program to support THALES Business Units in the evolution of their methodologies towards Model Driven Engineering (MDE). For that purpose, MIRROR supplies a generic MDE technology, which can be customised to specific needs. MIRROR, answering to Models Engineering, MDA and MDE stakes, focuses also on productivity improvement needs of THALES Business Units. The additional stake is that, by integrating additional reference principles, best practices and task automation (model validation, transformation of high level models to lower level models, generation of code or documents from models, automation of diagram generation, etc.), this leads to software development industrialisation.

The MIRROR program aims at defining a generic but customisable MDE approach and technology, including principles, methodology and tooling, that is suitable for most THALES Business Units needs in terms of system and software engineering. In that sense, MIRROR MDE puts more than MDA at work.

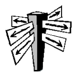
MDE in Practice

Putting MDE into practice requires addressing a number of inter-related dimensions:

- **Technical dimension**: support to model manipulation, model transformation, model merging, model configuration management, in terms of basic languages, mechanisms and tools.
- **Methodological dimension**: impact of MDE on the engineering processes, specification of models to be built at each engineering stage, specification of the information to represent in a given model, and of the way to represent it, models quality and validity control, models lifecycle, etc.
- **Know-how elicitation, building and formalisation aspects**: elicitation of the meaningful concepts for a given domain, capitalisation of engineering know-how and best practices into guidelines (e.g. What is a "good" analysis model? How to manage platform-independence in a design?) and into techniques for supporting or automating model building: specification of the engineering rules for automating some model refinement activities (for example, generating a CCM design out of a high-level design).
- **Organisational dimension**: impact of MDE on the software development organisation, identification of the new roles and activities, etc.
- **General technical change management issues**: how to introduce and develop MDE in an organisation, how to measure and control ROI, etc.

The MIRROR program addresses all of these inter-related aspects in a comprehensive way. In 2002, the main focus has been on the methodological, know-how and technical dimensions, leading to the constitution of a core technology.

The MIRROR core technology contains the following:

- **A reference set of MDE principles**, which establishes a common framework for model-driven engineering processes. These principles define requirements and recommendations for model-driven engineering processes in a Business Unit: the first-class role of models, the need for formalised modelling principles and rules, the definition of software engineering as a refinement process, the separation of concerns, the extensive usage of a number of standards, and the relationship with existing methods and process definitions.
-  **A MDE methodology that implements these MDE principles in a particular THALES context**: the context of *QoS-intensive middleware COTS-based software systems*, with a focus on *software development*. This MDE methodology takes the form of a chain of 5 models as the engineering backbone around which software development activities are organised. For each model of the chain, we define:
    - precise modelling principles and rules, a subset of which are currently being standardised at the OMG (QoS modelling rules)
    - process elements (roles and activities for building the model, related artefacts, lifecycle aspects)
    - rules for automating aspects of the model refinement activities, as well as code and configuration file generation (general engineering and CCM/EJB technological know-how), including also Platform Description Models (PDM) describing the services of the target platform and their qualities.

- rules for generating the standard THALES documents out of the models.

This methodology is for the main part tool-independent. The model manipulation techniques are however implemented within the Softeam/Objecteering tool.

- **Tooling** studies and developments at several levels, involving:
  - the clarification of tool requirements for MDE, as a support for choosing among existing market tools, in collaboration with the System CET.
  - the specification of a next generation of tools for MDE, in collaboration with the Softeam tool vendor.
  - the investigation of specific techniques for model transformation.

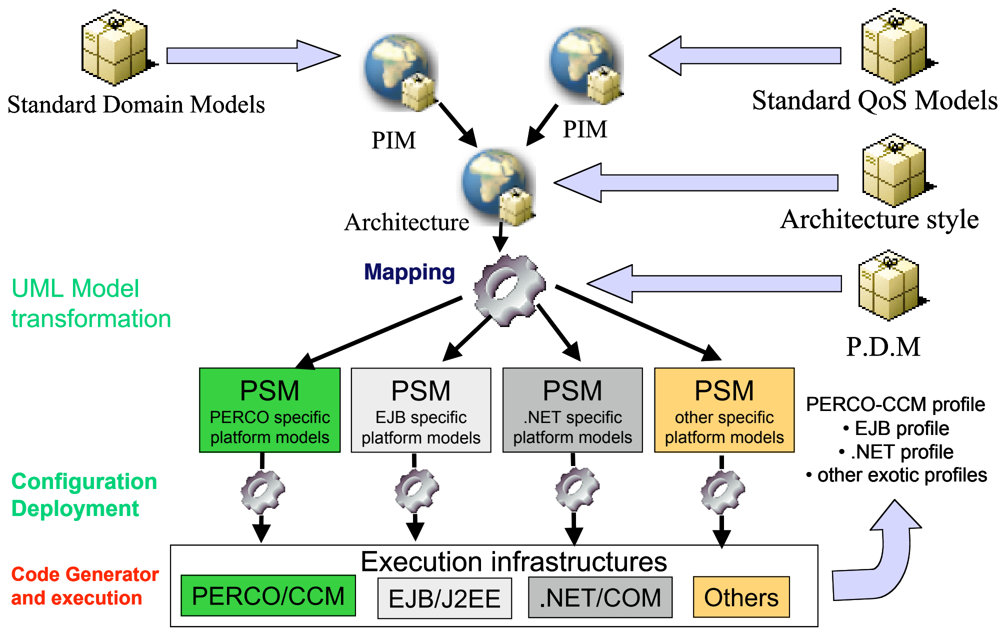


Figure : Global Picture of MIRROR MDE

Conclusion: Deployment at different levels

MIRROR organises, guides and supports software system development through:

- specifying *what to model, how and when*. MIRROR provides precise rules for using the UML and the related standards at the different stages of a project, from requirements formalisation to design and test.

  These rules address core THALES development concerns that are not well covered in the general literature such as the modelling of non functional requirements and qualities of services.
- specifying *validity, traceability and coherency rules* for controlling the meaningfulness, completion and quality state of the models.
- *improving the quality and maintainability of the designs* through defining an engineering approach based on the separation of concerns, and the controlled introduction of technological dependencies in a design. MIRROR separates domain-focused specification activities from high-level platform-independent design, and from platform-specific design, allowing to avoid technological pollution of a design, and to cope with

platform complexity.

- providing techniques for *automating some of the development activities*, in particular: the transition from a platform-independent design model to a CCM platform-specific model, CCM code and configuration file generation.
- *organising the development process* through defining well-focused roles, activities and tasks, as well as development products, in coherency with the THALES reference processes.

MIRROR provides support to process improvement people in a Business Unit for:

- assessing the interest of introducing MDE aspects in their processes,
- defining methodological MDE solutions based on the MIRROR core technology, adapted to their specific context (type of systems, technical requirements and constraints, industrial constraints, methodological background, technological baseline, etc.); this involves defining a model engineering chain that fully matches the Business Unit context.
- evolving the development environments and tools of the Business Unit so as to support the new approach to development.
- experimenting, assessing, consolidating and validating these solutions.
- introducing these solutions into the operational context through technical change management actions, involving training and coaching.

At the time of writing, MIRROR is engaged into practical process improvement activities within 3 main Business Units of THALES and proposes Model Driven Engineering approaches that address:

- *the improvement of productivity and quality* in software system development and maintenance, based on:
  - the capitalisation of domain, engineering and technological know-how within reusable models and techniques which provide means for actual reuse on projects,
  - the automation of aspects of the engineering work, based on capitalised engineering rules and technological know-how (automatic model refinement, code and configuration file generation, test generation),
  - the separation of concerns (domain vs. technology in particular), which leads to a separation of competencies and a better rationalization of engineering work, all the while improving the maintainability of designs and the evolutivity of products (better management of technological changes in particular),
  - traceability mechanisms that enable fine-grain evolution impact analysis.
- *a better technical control and mastering of products*, providing improved means for the assessments of costs at contract negotiation time. This better technical control relies on the availability of representations of the systems at different levels of abstraction, according to different viewpoints for complexity management sake, together with traceability and dependency links throughout the different dimensions of the software, from requirements down to the code.

MIRROR is gradually building practical knowledge and experience on the subject of Return On Investment, as an effective support to decision making for managers.

Experimenting the MDA approach in France Telecom

Mariano Belaunde, France Telecom R&D

May 28th 2003

Introduction

Since several years, object oriented modelling has being gained more and more acceptance within France Telecom. The UML notation is currently being used for multiple purposes, such as:

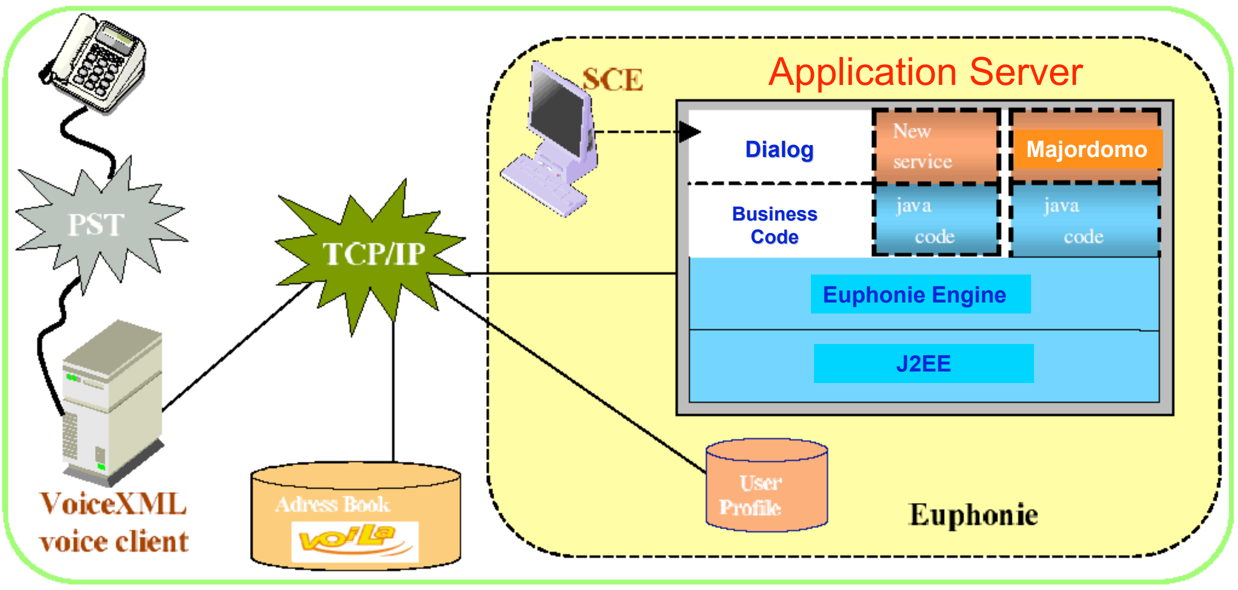
- To describe business process,
- To specify the structure of a telecom service,
- For analysis and design when developing new applications.

However, in almost all the applications that are currently being used within the information system of France Telecom, models only serve for documentation purpose, and are often obsolete in respect with the code they are supposed to specify. A significant effort is currently being done in France Telecom R&D to experiment scenarios in which the main parts of the business applications can be derived from models. The expected benefit from this would be the ability to built and integrate applications much more faster than in the past. However a number of organizational issues are raised when trying to change the standard development life-cycle. In this short paper we will describe two experiments that are currently being conducted in France Telecom:

A model-driven approach for telecom voice services based on VoiceXML.

This experimentation is conducted in the context of an European IST project named MODA-TEL. The objective is to produce two "real" applications, one running on top of the Telenor platform, the other running on top of the France Telecom platform. Both applications will share the same "high-level" description of the dialog (the PIM model of the dialog) however they will differ in the way they are mapped into proprietary technology and on the way VoiceXML is used.

France Telecom will use the EUPHONIE platform that he has developed internally. The main architecture is depicted in the figure below.



The methodology that is currently being defined for the purpose of the voice service use case will include at least the following steps:

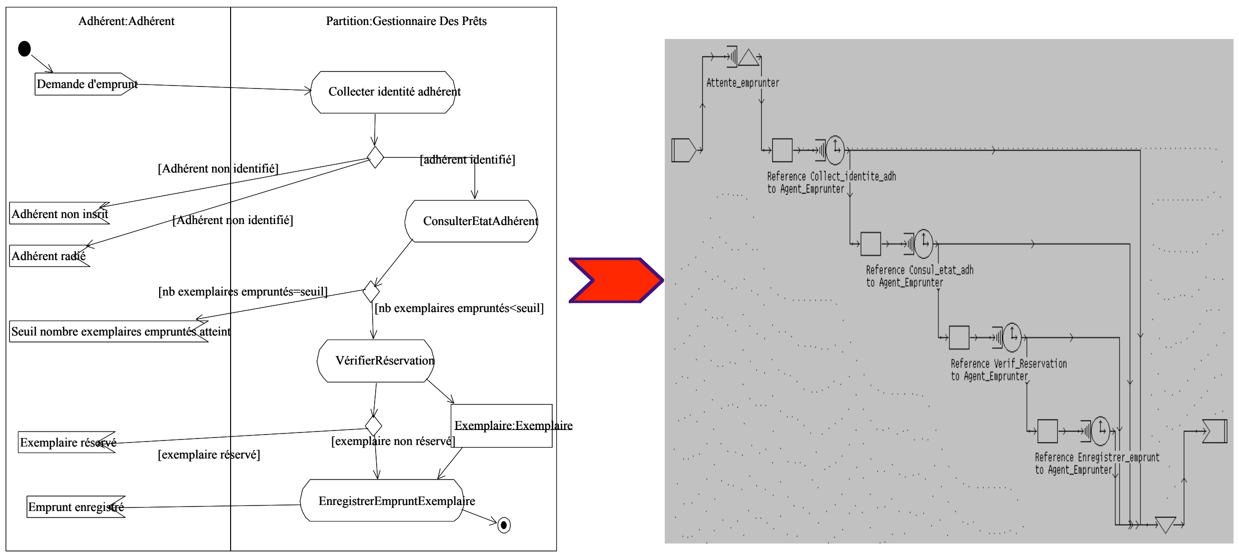
- PIM preparation phase: a PIM metamodel for the domain is defined. In addition to this two variants of UML profiles are defined: one compliant with UML 2.0 notation and another which uses the existing UML 1.3 capabilities. The behavioral specification of the dialog will be based on state machines paradigm.
- PIM modeling: A specific voice service, an address book facility accessed through a voice interface, will be specified in a platform independent way using the PIM UML profiles.
- PSM preparation phase: In the case of the France Telecom implementation, the PIM metamodel will be enriched with specific information dependant on the EUPHONIE platform. In the UML profile this will be reflected by the introduction of annotations.
- PSM modelling: The PIM models are annotated accordingly with the PSM enrichments defined in the metamodel and UML profile.
- Transformation specification: The mapping rules to produce the artefacts expected by the EUPHONIE platform will be specified here using natural language and an specific executable formalism named TRL – Transformation Rule Language – which was defined by the OpenQVT – in the nitial submission to the MOF Q/V/T RFP.
- Generation of the artefacts
- Manual coding for all the parts not covered by the specification.

Simulation of the voice service for quick testing is also an important issue that will be addressed. Since the tools used to edit the UML models and the tools for simulation are not the same an important requirement is the ability to export all the information captured in the UML and the ability to transform the models into another model that would be compliant with the target simulation tool.

Business process dimensioning.

The purpose of this experiment conducted internally at France telecom is to see how to configure in an optimal way the resources needed to run a business process. An example of business process in this case may be the process to sell ADSL packs to clients.

The main problem in this experiment is how to convert a UML activity-diagram based specification into one that will be compliant with the SES workbench tool for simulation and dimensioning. In addition an important task is to find the appropriate annotations that will bring the information missing in the initial business models, such as estimations on the average time spent in each activity, or the concurrency constraints that apply to an activity. The figure below shows an excerpt of the transformation.


Adhérent:Adhérent
Partition:Gestionnaire Des Prêts
Demande d'emprunt
Collecter identité adhérent
[Adhérent non identifié]
[adhérent identifié]
Adhérent non insrit
[Adhérent non identifié]
ConsulterEtatAdhérent
Adhérent radié
[nb exemplaires empruntés=seuil]
[nb exemplaires empruntés<seuil]
Seuil nombre exemplaires empruntés atteint
VérifierRéservation
Exemplaire:Exemplaire
Exemplaire réservé
[exemplaire non réservé]
[exemplaire réservé]
Emprunt enregistré
EnregistrerEmpruntExemplaire
Attente_emprunter
Reference Collect_identite_adh to Agent_Emprunter
Reference Consul_etat_adh to Agent_Emprunter
Reference Verif_Reservation to Agent_Emprunter
Reference Enregistrer_emprunt to Agent_Emprunter

Conclusion

We have described briefly two examples of experiments that are being conducted at France Telecom. These examples tend to show that each application case requires a specific methodology. The difficult aspect of the MDA seems not to be on the technical feasibility to grom from abstract descriptions to more concrete descriptions, but more on the managements aspects dealing with model versioning, traceability and so on.

MDA Experience and Deployment at Sodifrance

SODIFRANCE
4, rue du Château de l'Eraudière – BP 72438
44323 NANTES Cedex 3
FRANCE
Phone: +33 (0)2.40.18.52.03
contact@mia-software.com
www.sodifrance.com www.mia-software.com

Sodifrance and the MDA initiative

Sodifrance is an IT services company in full expansion. Headquartered in Rennes (France), Sodifrance employs over 900 employees in 14 locations in France and Benelux. Sodifrance offers a wide range of activities (engineering, outsourcing, integration, migration and consulting). Sodifrance also develops and distributes software solutions allowing the "industrial" development of complex applications and information systems and the evolution of legacy systems towards new systems or new technologies.

The efforts engaged during the last 10 years by Sodifrance in R&D, through collaborations with academic and industrial partners, have lead in the development of a wide set of solutions dedicated to the evolution of information systems : Essor (information system analysis and documentation), Semantor (code analysis) and, more recently, **Model-In-Action** (model transformation and code generation).

These solutions are all based on meta-modelling techniques for representing, querying and transforming data. It is therefore quite natural for Sodifrance to participate within the MDA initiative as a tool vendor, as a practitioner and as a contributor (Sodifrance is member of the **OpenQVT** consortium submitting a proposal for the OMG QVT RFP).

Model-In-Action

Model-In-Action is a tool suite that implements OMG's MDA concepts. The suite is composed of the two major tools :

- **MIA-*Transformation*** : to build Model-to-Model Translators.
- **MIA-*Generation*** : to build Model-to-Code Translators.

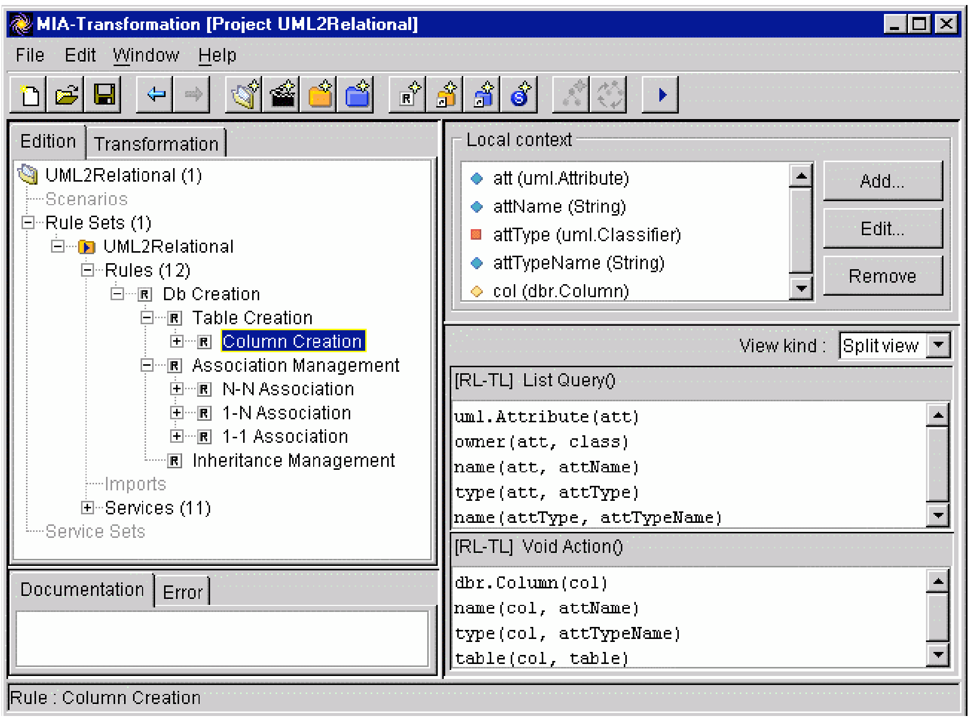
Model-In-Action supports OMG's standards :

- The **MOF** (Meta-Object Facility) to define metamodels
- **XMI** (XML Metadata Interchanage) to exchange models with CASE tools
- MIA-*Generation* supports **UML Profiles**

MIA-*Transformation*

MIA-*Transformation* performs transformations from source models into target models, regardless the input and output formats (XMI, other file format, API, repository). Transformations are expressed using inference rules, which may be enriched using Java scripts for additional services such as string handling.

The MIA-*Transformation* environment provides a graphical editor to specify the rules and a built-in generator of MOF components used to manipulate source and target models.



```
uml.Attribute(att)
owner(att, class)
name(att, attName)
type(att, attType)
name(attType, attTypeName)
```

```
dbr.Column(col)
name(col, attName)
type(col, attTypeName)
table(col, table)
```

Figure 1. MIA-Transformation graphical editor

MIA -*Transformation* has been used in various domains, such as:

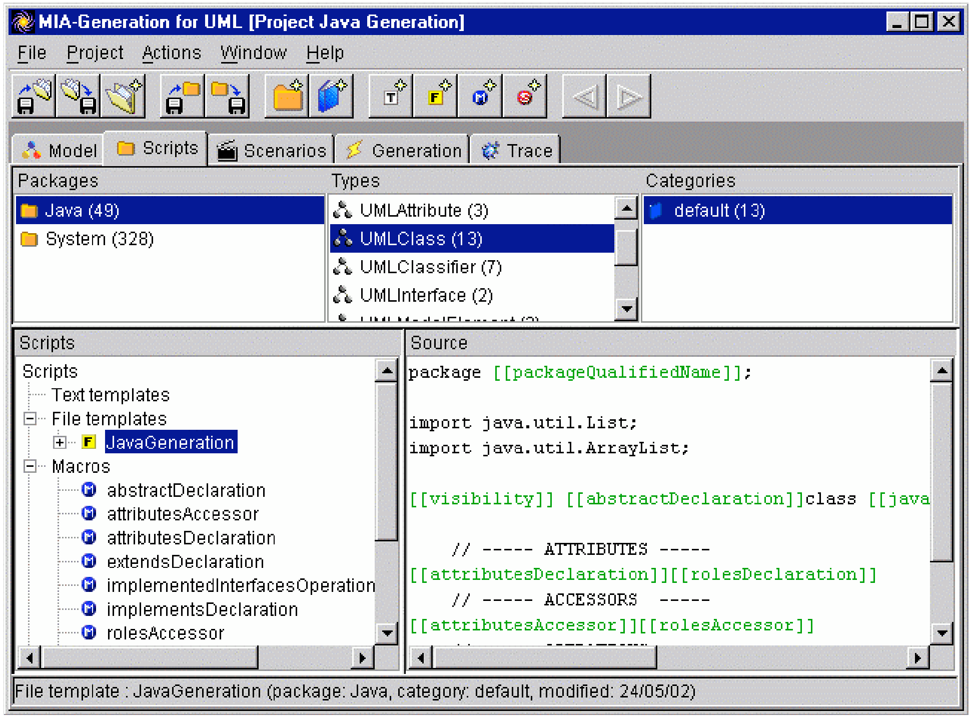
- Transformations between system engineering tools (Statemate™ and Core™) and UML models.
- Transformations between graphical user interface models and UML models.
- Transformations of process models to workflow engines.
- Split or merge of models.
- Migration of client-server applications to J2EE applications

MIA-*Generation*

MIA-*Generation* is a development environment for custom generators. MIA-*Generation* is open to any file format for input models and can be connected to any major modeling tools (Rose™, Rhapsody™, Together™, Argo, and any XMI compliant tool).

Generation rules are specified with a full IDE dedicated to generator building using:

- Templates (WYSIWYG scripts), which let you enter the text to be generated instead of programming (e.g. copy/paste of best code samples).
- Java™ is used as scripting language for more complex operations to avoid the need to learn a proprietary language and to allow using any java components.



Packages	Types	Categories
Java (49)	UMLAttribute (3)	default (13)
System (328)	UMLClass (13)	
	UMLClassifier (7)	
	UMLInterface (2)	

Scripts
- Text templates
- File templates
  - JavaGeneration
- Macros
  - abstractDeclaration
  - attributesAccessor
  - attributesDeclaration
  - extendsDeclaration
  - implementedInterfacesOperation
  - implementsDeclaration
  - rolesAccessor

Source

```
package [[packageQualifiedName]];

import java.util.List;
import java.util.ArrayList;

[[visibility]] [[abstractDeclaration]]class [[java

    // ----- ATTRIBUTES -----
[[attributesDeclaration]][[rolesDeclaration]]
    // ----- ACCESSORS  -----
[[attributesAccessor]][[rolesAccessor]]
```

File template : JavaGeneration (package: Java, category: default, modified: 24/05/02)

Figure 2. MIA-Generation graphical editor

MIA-*Generation* is used to build code generators complying with specific architectures and frameworks. MIA provides a dynamic bridge between analysis and design, and supports iterative cycles by preserving user defined code fragments between two generations.

Effective deployment of MDA

Model-In-Action supports **MDE** (Model Driven Engineering). In an Y-cycle development process, it allows for transforming PIMs into PSMs, by integrating framework-related concerns, and for producing automatically the application code from PSMs.

Model-In-Action has already been used by several clients such as BNP Paribas, Credit Agricole, Banque Populaire, CCF, Crédit du Nord, MAAF, Vinci (for generating J2EE code), Thales (C, C++, Ada), France Telecom Equant (ASP, VB, XML) and Diehl Avionics. Model-In-Action is used by Accenture to industrialize code production compliant with their framework. Model-In-Action technology is also embedded within partner's solutions such as Rhapsody (Sodius/I-Logix) and W4Studio (W4).

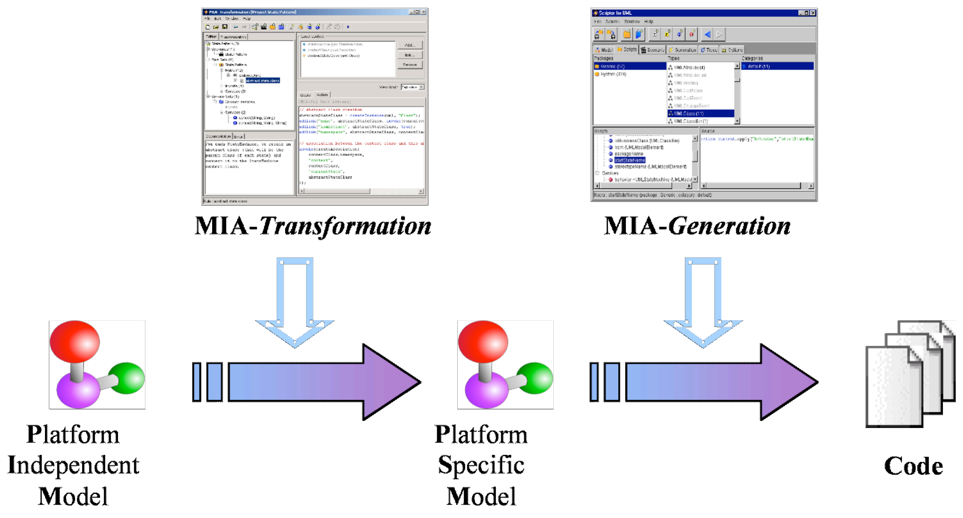


Figure 3. MDA-based software development process using Model-In-Action

Model-In-Action may also be used for transforming legacy applications. Sodifrance has developed the following process :

- Reverse-engineering of the legacy system into a reverse-engineering model
- Transformation from the reverse-engineering model into a target system one
- Generation of the new application from the target sytem model

The production of the target system model from the reverse engineering model may be done either in a straightforward way or using intermediate models. A major benefit of this approach is that the model of the target application is produced and may be used for further evolutions. Such an approach has already been used on industrial projects :

- MAAF : migration from NSDK to UML and JSP / Java / Struts
- Crédit du Nord : migration from Cool:Gen to UML and JSP / Java / XML

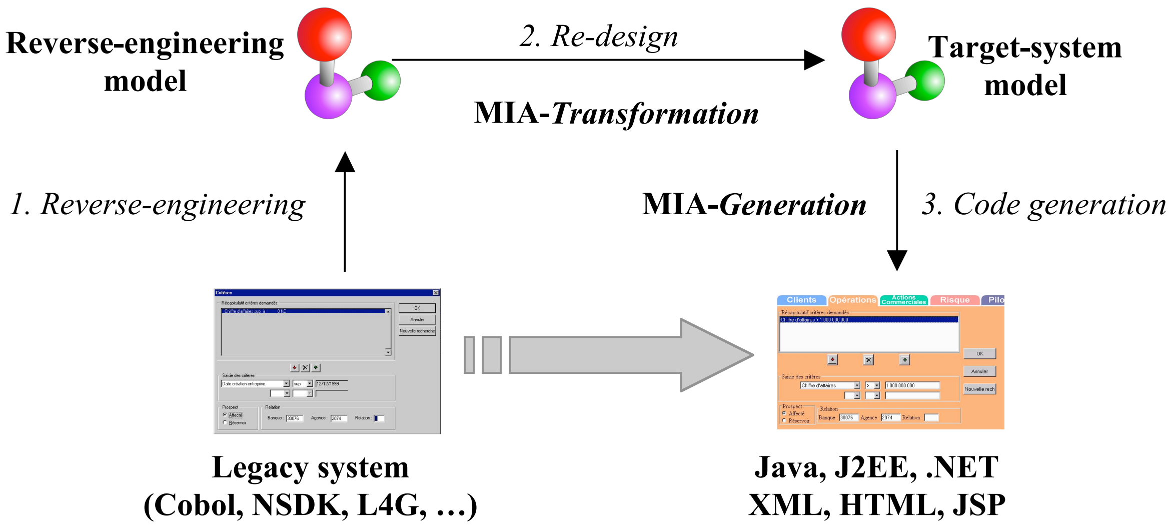


Figure 4. Legacy transformation process using Model-In-Action

OpenTool:
How does a metatool approach fit the needs of MDA and model-driven engineering

Gilles m. Pitette

+33 (0)1 53 38 46 00
Gilles.Pitette@tni-valiosys.com

174-178 Quai de Jemmapes
F-75010 Paris
www.tni-valiosys.com

Models, MDA, ... model engineering languages and tools

Building models to describe a system, be it software-intensive or not, is not a new idea. Models of a system may be of many different natures, serving different purposes in the engineering process. Models may aim at describing the structure of a system, possibly from different perspectives, thus providing a mean to organize and guide its construction. Models may also be used for expressing what a system should be, or allowing its simulation in order to validate its definition before initiating its construction process.

The software engineering literature usually presents visual modeling as a mean for managing the complexity of a software system. Models represent its design by providing abstractions at multiple levels, with multiple views, each addressing the needs of specific stakeholders in the development process. From that perspective, visual modeling is an essential communication vehicle across the various actors of this process. Over years, various approaches to analysis and design have lead to the definition, and usage, of a multitude of visual modeling notations.

Models "unifying" the development team must be based on *a modeling language that is shared by all of model users*. Such a language is thus constituted of two equally important parts:

?? a set of common concepts pertaining to the domain being modeled, together with a structure to combine them into models: this is often termed as the abstract syntax, or *metamodel*, of the modeling language,

?? a visual representation of these concepts and constructs: this provides the concrete *graphical syntax* used to author and publish models.

Models are actual work products in the engineering process, as well as assets for the organization owning a system. It is therefore essential to provide such an organization with means to conveniently author and edit models, store and manage them, as well as publish them through consultation mechanisms appropriate to their various readers. Further, the availability of models opens the possibility of exploiting them in various ways, e.g. for generating code implementing them.

Until recently, engineering approaches were mainly document-driven. Analysis and design models have been used mostly for documenting the development a software system, while source codes remained the primary assets constituting the capital of an organization depending on software. *With the advent of the Model-Driven Architecture (MDA) models become the first-class citizens*, defining concretely the system(s) on which organizations business rely. With the MDA , the software industry moves towards a model-intensive industry where artifacts such as source code become secondary, being dependent on specific system execution platforms and derived from platform independent models, possibly in an automated manner.

Given the importance of models in that context, two aspects become essential:

1. *The language(s) used for modeling must fit at best the subject domain* being addressed by each model in a model-driven engineering process. This is true not only for the domain of system being developed, but also for the specific role of each model handled in the engineering process.
2. *Appropriate tools and automation must be provided to engineer models*. Their ability to build, manipulate and exploit the variety of models involved in a model-driven engineering process must be in balance with the importance of models within the MDA.

What are the implications of MDA

With the model-driven architecture and engineering approach, models that are as much as possible independent of the system execution platform (PIMs) are built and then progressively evolved to models that include features more specific to such platform (PSMs). Adopting MDA has a number of implications. Some of them call for a specific support from modeling tools.

“Efficient” models require specialized modeling languages

In that context, different kinds of models are developed, with a variety of modeling needs. Therefore the need for specialized modeling languages (metamodels) rapidly emerges.

For all promises of MDA to be achieved, all modeling efforts must be “efficient”, i.e. models must be of “high quality”. This means for example that building a PIM requires a modeling language that supports naturally, and cleanly, the concepts and constructs of the subject domain. In the same way, high quality models using concepts specific to certain techniques, architectures or technologies must be expressed by the modeling language supporting models using them.

Initially, the Unified Modeling Language (UML) was designed as the unification of object-oriented analysis and design methods for software systems. The UML may be extended with profiles to provide representations of concepts needed in a certain modeling context. This is done mainly by associating “stereotypes” to UML metaclasses. However, experience has shown that UML profiles is a too limited extension facility to fit certain modeling needs. Further, “extending” a language makes it more complex, while some of its existing constructs become useless for the subject area. Example of this may be found in some attempts to adapt the UML and use it in new domains that could also profit from the MDA. The UML 2.0 effort has recognized this need for more flexibility. Beyond dalects of UML, defined using profiles, it introduces the concept of language in the UML family, defined with a metamodeling facility reusing a common infrastructure.

Model transformation is essential

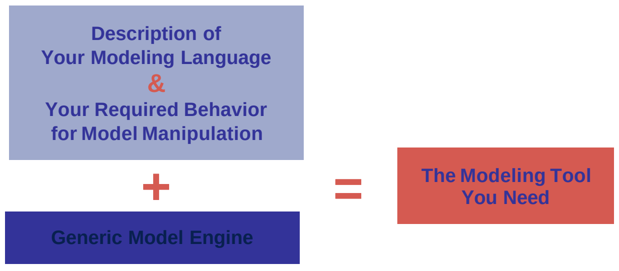
Model transformation is inherent to the MDA. The source and target models of a transformation may be the same or different. In the latter case, transformations must be guided by the expression of mappings between source and target metamodels. All these aspects must be automated as much as possible.

The user should be guided

The MDA describes a general approach where models play a central role, and are evolved from PIMs to PSMs. Processes adopted by organizations or projects applying the MDA have to be considered consequently. A process usually defines who does what, when, and how. It describes a set of activities with roles, input and output artifacts, tasks and supporting tools. With the MDA, the artifacts should be qualified by the metamodel that governs their structure. In the same way, each activity or task should be associated with a tool (or a set of tools) capable of handling the metamodels of the concerned artifacts, as well as applying the relevant and automated transformations. More, when enacting the process, it is desirable that the user executing a task be as much as possible guided by its tooling environment.

What is OpenTool

The metatooling approach



OpenTool is essentially a configurable toolset that supports the creation of model engineering tools thanks to a metamodeling approach.

It is architectured around a generic model engine. A model manipulation tool specific to a given modeling language may be obtained by associating a configuration to this model engine.

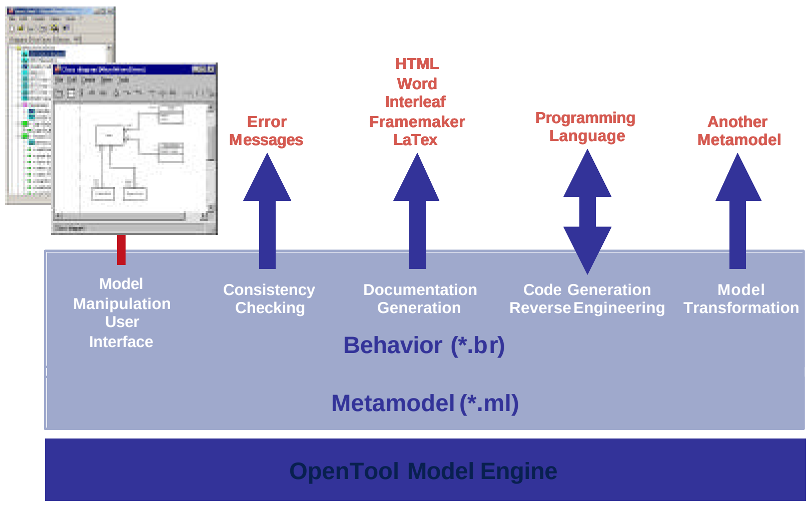
A configuration not only describes the modeling language to be supported by the desired tool, but also the defines the functionality required for manipulating models.

The OTScript metalanguage

An OpenTool configuration is fully programmed using the OTScript language, and is exploited by a generic model engine. Each configuration is made of into two different parts:

?? the metamodel which defines the abstract syntax for models,

?? the behavior associated to metaclasses which implement the tool desired functionality for manipulating a model.



The metamodel of the supported modeling language is coded in a set of ML files. For doing so, OTScript offers the usual object-oriented concepts (-meta-class, inheritance, attributes), but also a concept of bi-directional association between (meta)classes. One may also generate a complete metamodel from UML class diagrams.

The behavior associated to (meta)classes is coded in a set of BR files. Behavior can be of 3 different kinds:

Graphical user interface OtScript offers a number of concepts and mechanisms to fully program a graphical user interface for authoring, displaying and editing models. Predefined constructs allow to easily code model *browsers* (be they usual tree-like browsers or different ones), *diagram editors* (e.g. such as editors for UML diagrams), *property sheets* for model elements, as well as free windows.

Consistency checking rules An OTScript rule define constraints that must be satisfied by any instance of the associated metaclass in a model. The OpenTool engine exploits directly these rules for checking a model. For example, any model element breaking a consistency rule automatically has a special appearance everywhere it is displayed in the associated GUI. Consistency rules may also contain explain, and repair, clauses for explaining the context of the broken constraints, and repairing the model whenever it is possible.

Methods As in any OO language, methods may be associated to OTScript (meta)classes. They may be used for navigating, querying and modifying the model as well as to interact with the model engine and the external world.

In addition to being object-oriented, OTScript is specially designed for manipulating models. It is thus:

Navigation-oriented The web of elements in a model is navigated through the associations natively defined in the metamodel with OTScript using a simple dotted notation.

Set-oriented Any expression in OTScript results in a set of elements in the model being manipulated. Language constructs, such as access to attribute or method call, are always implicitly applied iteratively to all the elements contained in the set subject to them.

Statically type checked OTScript is a compiled language. Types are fully checked at compile type, so no typing error may be raised at execution time. Compilation produces a kind of byte-code that is executed by the OpenTool engine.

All these characteristics make OTScript a highly expressive and efficient language to code behavior such as

Code generation	(and reverse engineering) to (from) a given programming language.
Documentation generation	A dedicated framework is provided to address usual document formats
Model transformation	A configuration can include both source and target metamodels.

Modularity for highly configurable tools

The definition of an OpenTool configuration is modular under two different perspectives. First, with OTScript, as opposed to usual object-oriented languages, the definition of the (meta)classes (with their inheritance relationships, attributes and associations), is physically separated from the definition of the behavior (methods, rules, user interface elements) attached to them. Second, each of those two subsets of an OpenTool configuration, metamodel and behavior, may be distributed across a number of modules.

A metamodel module is coded in a separate ML file (with ".ml" file name extension). Each metamodel module may of course introduce new (meta)classes, but also extend the definition of a (meta)class with new attributes and associations.

A behavior module is coded in a separate BR file (with ".br" file name extension). Each behavior module may contain any piece of behavior. The definition of each single method, rule or user interface elements states to which (meta)class it is associated.

An OpenTool configuration simply provides the list of all the modules, metamodel ones and behavior ones, that are merged to constitute it. All the modules of a configuration are loaded when OpenTool is launched to provide the modeling tool that is specifically needed. Of course, the consistency of such a configuration is automatically checked. From that point, the desired tool runs, driven by the appropriate metamodel and providing the required model engineering facility.

Modeling tools obtained with OpenTool are thus highly configurable. Depending on the care taken to organize metamodels into the appropriate modules they can be configured to support meaningful subsets, or supersets, of standardized metamodels. Also, depending on the organization of behavior modules, functionality provided by modeling tools constructed with OpenTool may be configurable up to a very fine grain level. It should be noticed also that an OpenTool configuration may result in a tool, or tool component with almost no, or very few, graphical user interface. As mentioned earlier, it is possible to have several different metamodels in an OpenTool configuration. This is the case for model transformation tools built using OpenTool.

An OpenTool configuration forms a graph of modules merged together. Such organization is quite similar to the one defined by the "merge" relationship between packages architecturing the UML 2.0 metamodel as currently proposed. However, in contrast to packages in the UML 2.0 metamodel, modules in an OpenTool configuration may associate operations (OTScript methods), and other behavioral items, to a (meta)class. Consequently, an OpenTool configuration may be represented package diagrams related by "merge" relationships, and generated from them.

How does OpenTool supports MDA and model-driven engineering

Many of the characteristics of OpenTool exposed in the previous paragraphs make it very suitable for addressing the needs of a model-driven engineering approach to development of software systems. This is likely applicable also to systems not only constituted of software.

The following discussion will be conducted in the light of MDA implications discussed earlier.

"Efficient" models require specialized modeling languages

OpenTool obviously provides support to the use of specialized modeling languages for addressing the modeling needs created by various domains, techniques, architectures or technologies. Not only OpenTool allows to define the metamodel of the required modeling languages, but also it provides means to easily implement tools including features to author models, edit, check, publish, document, transform, or implement them in programming language code.

Specialized modeling languages may either be fully defined or derived from existing ones by profiling. Profiling is just a constrained form of metamodeling. With OpenTool, a profile results in a (small set of) module(s). A profile (meta)modeling tool has already been developed using OpenTool. From the diagrams defining a profile in UML 2.0, with the "extension" relationships between stereotypes and metaclasses, it generates not only the corresponding OTScript metamodel modules, but also the behavior modules providing the graphical user interface to handle the stereotypes. Further, managing stereotypes in a resulting tool is fully similar to the management of any other module.

Model transformation is essential

Designed especially for that purpose, OTScript is a very powerful language to manipulate models. This is why it is navigation-oriented and set-oriented. More, OTScript is a language that is completely independent of the metamodel that structures the subject model. In addition, several metamodels may be combined in an OpenTool configuration. From our perspective, such a linguistic approach (a specialized language for model manipulation) is much more suitable than manipulating models using a general-purpose language and a set of APIs (one per metamodel) to access models. Model transformation programs can thus be easily and efficiently developed with OTScript.

However, there is a need for defining a standard language for describing mappings between metamodels and transformation of models. TNI-Valiosys, as a partner in one of the current submissions, is actively involved in the process initiated by the OMG via the MOF QVT request for proposals for addressing this issue. Our expectations are that programs written in the “QVT language” will be easily implementable with (translated to?) OTScripts programs.

Adopting such an approach, constructing a tool providing ways to edit a model with metamodel S and transform it to a model with metamodel T, is simply done by providing an OpenTool configuration including:

1. modules defining the source metamodel S, together with behavior modules providing facilities to edit a model,
2. modules defining the target metamodel T,
3. modules providing the transformation behavior, be they coded directly in OTScript or resulting from a translation of program written with the QVT language.

The user should be guided

The MDA describes a general approach where models play a central role, and are evolved from PIMs to PSMs. Processes adopted by organizations or projects applying the MDA have to be considered consequently. A process usually defines who does what, when, and how. It describes a set of activities with roles, input and output artifacts, tasks and supporting tools. With the MDA, the artifacts should be qualified by the metamodel that governs their structure. In the same way, each activity or task should be associated with a tool (or a set of tools) capable of handling the metamodels of the concerned artifacts, as well as applying the relevant and automated transformations. More, when enacting the process, it is desirable that the user executing a task be as much as possible guided by its tooling environment.

The importance of defining a development process is equally important with a model engineering approach as with a “traditional” development approach. OpenTool provides an ideal means to develop a process definition tool, for example using the metamodel proposed by the SPEM standard defined by the OMG. Doing so, model-engineering process may be described themselves as models. In addition, the description of an activity, or task, in such model-engineering process models should include the identification of input and output metamodels. These process models can also be automatically transformed to project management models or workflow models, and be processed by existing specialized tools when enacting a process.

However, further remains a need for tools to guide the end user when performing an activity in the process. For that purpose, with OpenTool a specific tool may be configured very easily to fit exactly the needs of an activity in a model-engineering process in terms of metamodels involved and model manipulation behavior. This may be done down to a fine grain level. For a given activity in a process the configuration may select:

?? subsets of the input and output metamodels relevant for the activity,

?? only those consistency rules and diagram editors relevant for the activity, or even consistency rules specific for the activity,

?? etc.

As an example, a tool configuration for supporting an analysis activity using UML shall include:

1. only those modules of the UML metamodel relevant for conducting an analysis,
2. only those behavior modules supporting UML diagrams relevant for conducting an analysis,

3. only those behavior modules containing consistency rules relevant at the analysis level where a lot of things are left undefined, or conversely specific rules applying to analysis classes,
4. behavior for transforming an analysis model to prepare it for the design activity.

Similarly, a tool configuration for supporting an design activity using UML for a real-time system shall include:

1. those modules of the UML metamodel relevant for conducting a real-time design,
2. behavior modules supporting UML diagrams relevant for conducting a real-time design,
3. behavior modules containing consistency rules relevant and specific to a real-time design,
4. modules supporting the relevant profiles, e.g. SPT and QoS,
5. behavior for generating a model for input into an RMA tool.

Conclusion

Developing models and modeling language is not new to software and system engineering. However with MDA models become very crucial assets. Model-driven engineering requires both good modeling languages and good tools to automate their usage.

We have shown in this paper how OpenTool provides essential features for supporting the various needs implicated by the model-driven engineering processes advocated by the adoption of the MDA.

Putting the MDA to Work

by Jean-Philippe LERAT – SODIUS – jplerat@sodius.com

Introduction

SODIUS started as French company dedicated to systems engineering and modelling.

After a couple of years and due to the new opportunities given by the MOF, XMI and the MDA approaches, SODIUS decided to complement its activities to broaden the use of the models made at a system level. As systems engineers spend a lot of money in creating and validating models for system-level specifications, it is a shame to limit the use of these models to produce only paper-based component-level specifications. Our vision is to foster the reusability of these models and optimize their value outside the strict systems engineering domain.

From a technical point of view, we started by using the MDA-oriented tools designed by the SODIFRANCE company (the Model-in-Action technologies) and complemented them with specialized developments, tools and applications, to form a complete, solid and consistent model-driven workbench.

On the road to this objective, several intermediate products are currently available for the users, all using the MDA. The following projects show how MDA was used in various projects as well as the added-value given by the MDA.

- a Statemate® to UML bridge
- a seamless system/software process for CORE®
- a design environment for hardware (FPGA)
- a UML environment for Ada programmers
- a graphical layout generator for C4ISR

What is the added value of MDA for our purpose?

“Software buses” and “multi-point software communication” are not new in the world. In the last twenty years, we have seen many promising projects in this arena. The achievements have always been under expectations and only a few of the promises have been satisfied.

For a long time, the focus was on the “plumbing side” of the problem: how to upload and download data within the databases of different software tools. In this respect, many standards have been made to carry information in and out of databases. Whether they are named SQL, COM-API or STEP, all of them share the same spirit – they are data-centric rather than concept-centric.

All of the different projects failed because this area of work is much too narrow for achieving the purpose: where projects aimed to link processes, the tools linked only data; where goals was to tackle semantic ruptures, tools addressed only low-level fields-and-column mapping.

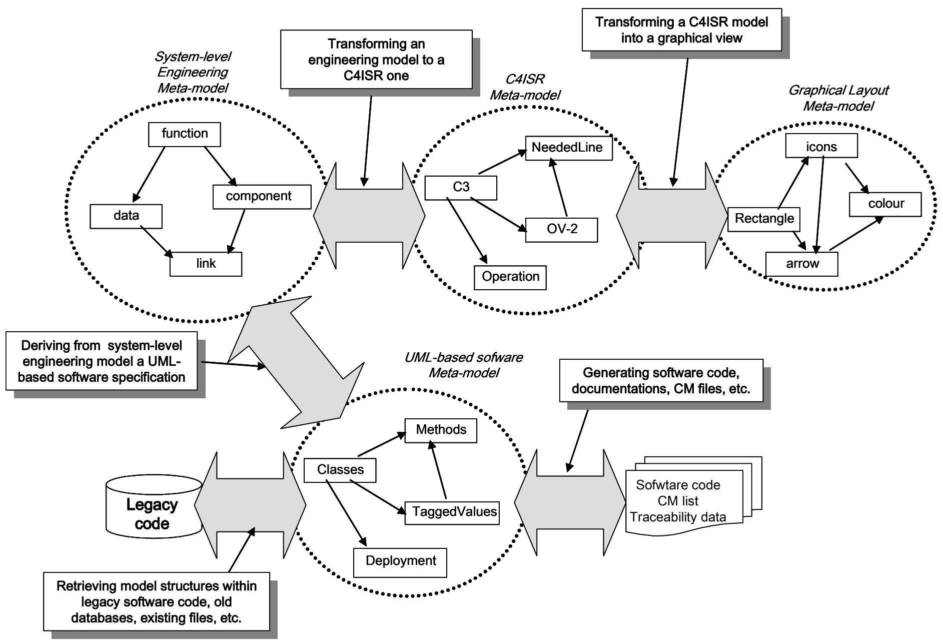
At this level, the software engineers had the duty to specify the mapping rules as well as the use cases for those kinds of bridges. Since the mapping rules and use cases are tied to the user’s processes, it was likely to have large discrepancies between the solution achieved by IT departments and the expectations of the users. However, it was difficult to include users in these specifications, since the IT department quickly faces software-related problems to establish parsers, data-type transformers, database accessors, etc.

The MDA approach provides tools for ***semantic engineering***: the raw material is no longer a subset of data processing concepts, but a bag of semantic primitives representing the concepts understood at the user level. This bag is complemented with low-level layers that are devoted to handle data and interface them with the semantic layer.

At this stage, the MDA brings a keystone to solve the problems: a way to clearly separate the process to process specification and the low-level data handling layer.

This is a key success factor: MDA helps to bring together different stakeholders and facilitate their relationships: the users can more easily describe their process-to-process mapping, while software engineers focus on the low-level aspects. In several cases, it was even necessary to break this communication into several steps, installing articulations in the overall process, making it easy-to-design, easy-to-maintain, and improvement-capable.

For instance, the figure below shows how we link a systems engineering tool, like CORE®, to achieve on one hand a C4ISR view that is translated into a graphical layout and, on another hand, a software specification to be used by the software people. Without the support of the MDA and its semantic skills, this achievement would have been much more complicated to get and the achievement would have been a rigid process, unable to support any further evolution.


System-level Engineering Meta-model
function
data
component
link
Transforming an engineering model to a C4ISR one
C4ISR Meta-model
NeededLine
C3
OV-2
Operation
Transforming a C4ISR model into a graphical view
Graphical Layout Meta-model
icons
colour
Rectangle
arrow
Deriving from system-level engineering model a UML-based software specification
UML-based sofware Meta-model
Methods
Classes
TaggedValues
Deployment
Generating software code, documentations, CM files, etc.
Sofwtare code
CM list
Traceability data
Legacy code
Retrieving model structures within legacy software code, old databases, existing files, etc.

The Statemate® to UML Bridge

The purpose of the Statemate to UML Bridge was to feed UML-based software processes with information automatically derived from Statemate models.

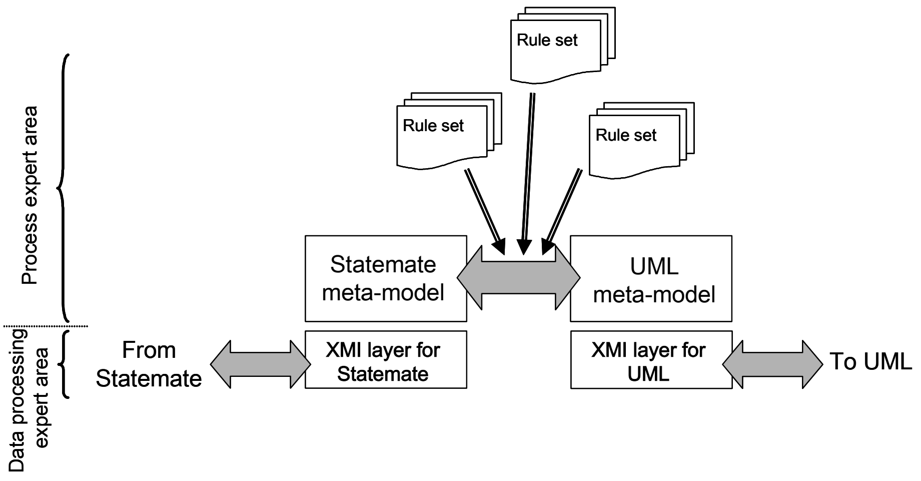
In addition, it works in the other direction to feed statemate models with UML-based information, like use cases and activities, made by the system-level layer and being the source requirements of the specification for a product being engineered with Statemate.

To achieve this bridge, we have created a metamodel of Statemate, using a MOF-compliant model. When this metamodel was created, an XMI reader and writer have been generated using Novosoft's open source tools.

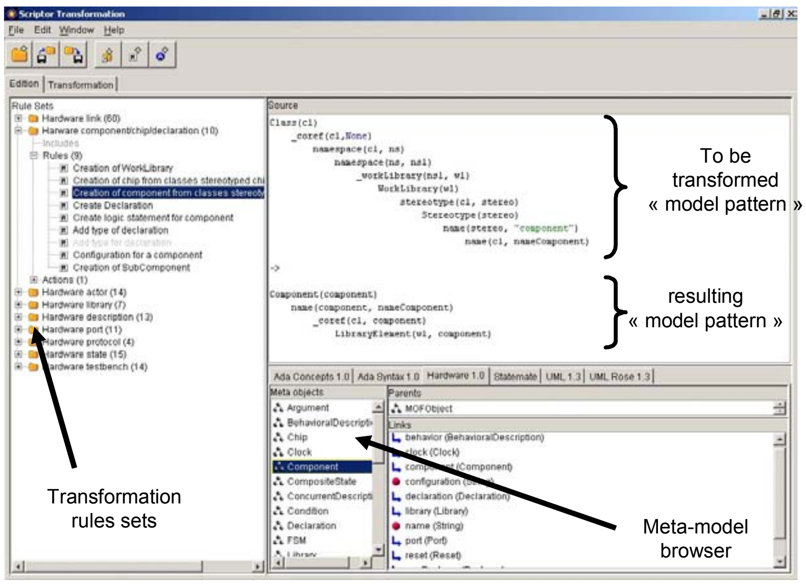
At this stage, the Statemate metamodel can be introduced in the different tools of our MD Workbench to generate code and participate in model transformation.

The other metamodel used is the UML 1.3 model provided by the OMG.

Model transformation is made by applying rules defined on the metamodels. Each rule set provides a transformation which creates objects in the target model based on patterns found in the source model.



The capability to transform models based on rules specified at the metamodel level is accomplished by using the Scriptor Transformation tool (a predecessor to Model-in-Action Transformation).



```
Class(cl)
    _coref(cl,None)
        namespace(cl, ns)
            namespace(ns, nsl)
                _workLibrary(nsl, wl)
                    WorkLibrary(wl)
                        stereotype(cl, stereo)
                            Stereotype(stereo)
                                name(stereo, "component")
                                    name(cl, nameComponent)

->

Component(component)
    name(component, nameComponent)
        _coref(cl, component)
            LibraryElement(wl, component)
```

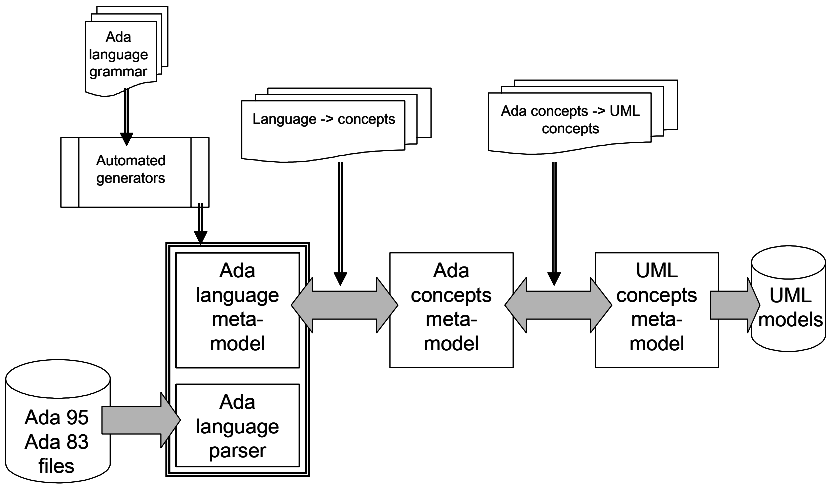
UML environment for Ada programmers

When the use of the Ada programming language started to decline in the 90's, UML was not the unavoidable graphical notation that it became. At the same time, winning UML companies of today do not envision to invest on the declining Ada market. The current situation leaves Ada programmers with poor solutions for Ada generation, most of them based on cryptic scripting languages dealing with class diagrams only. MDA was used to provide a cost-effective solution to bridge the Ada programming language with the UML graphical notation. This was achieved by associating code generation templates to the meta-types of the UML metamodel, which then work together to generate a complete Ada program.

But beyond the generation of Ada from UML diagrams, a major requirement for the solution was to enable a migration path from the Ada environment to other languages. SODIUS used the MDA to produce a rule-based reverse engineering solution which offers a flexible way to tailor the instantiation of UML models from Ada code.

Rather than trying to create UML elements from a parse tree, which is cumbersome and difficult to achieve, SODIUS has developed MDA-oriented tools to generate MOF-compliant meta-models from a BNF-style grammar description. Then, different steps of rule-based transformations between the various metamodels ease the capture of the requirements for reverse engineering and their implementation.

The general architecture of this technology available for I-LOGIX's Rhapsody is shown as below:



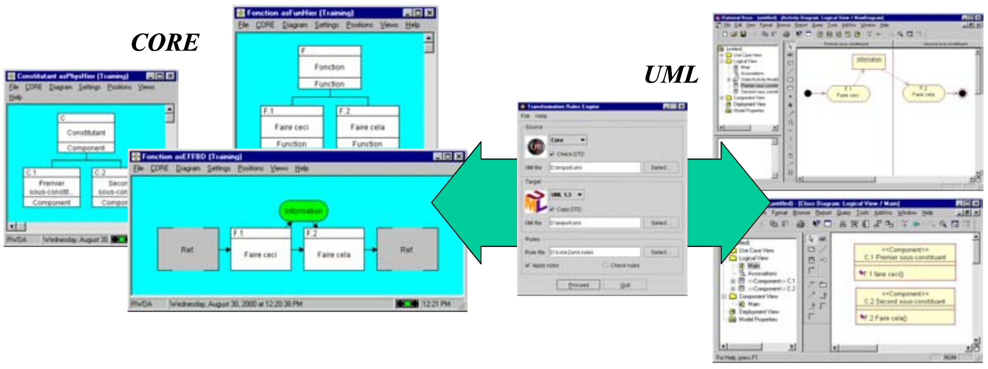
The effect of such architecture is that the expertise needed to build and maintain the engine is clearly separated. The rules concerning the mapping of the syntax elements for a language to the conceptual view of the language are separated in the conversion from the Ada syntax metamodel to the Ada concept metamodel, and the rules for mapping Ada concepts to UML concepts and kept in the conversion from the Ada syntax metamodel to UML. Another advantage is that different projects or users can employ different mappings from Ada to UML without impacting the entire architecture.

Seamless system/software process for CORE®

The aim of this project, which is still under development, is to reduce the time and cost of the system-to-software communication. As UML has shown limitations when designing high-level systems which involve human beings, hardware, spacecraft, ground segments, etc…, specialized tools for these types of applications ,like CORE®, do not clearly interface with the other disciplines that implement the component-level, like software engineering.

When paper-based system-level specifications are given to software engineers, they first start by reading the document and translating it into UML concepts and diagrams. This step costs a lot of money, wastes a lot of time and introduces the need for a validation activity that reviews the UML-flavoured spec to make sure that it does not betray the originating specification.

The goal here is to generate automatically a UML model that derives from the paper-based specification, saving the time, money and risk of the manual translation.



In this case, all CORE's components are translated into a stereotyped class, and all functions allocated to a component are considered as a method of the class. The behaviour models of CORE are translated into activity diagrams.

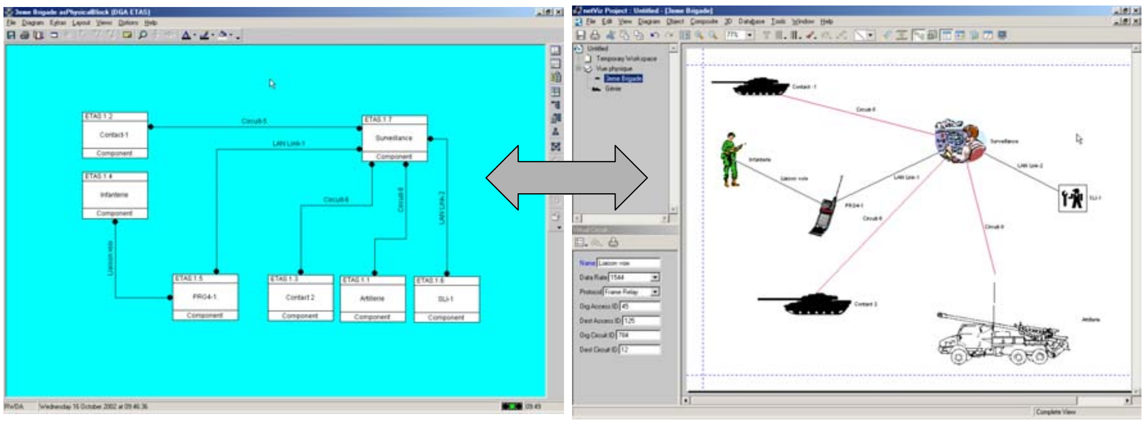
More is being added, such as the translation of the system-level behaviour models into sets of UML use cases for the modules to be implemented.

Graphical layout generator for C4ISR

C4ISR proposes a specialized metamodel, defined by the Department of Defense within the various C4ISR architecture frameworks.

This metamodel includes concepts for systems engineering, operational description, integration modelling, etc. The models shall be presented using specialized tables and defined views. Obviously, these views do not exist as such in many engineering tools.

Rather than adding these views into existing tools, we are currently designing for the NetVIZ® product an XMI bridge that will provide a semantic level interface between its metamodel, composed of icons, boxes, arrows, rectangles and tagged value-like fields, and the metamodel of engineering applications like CORE, UML modellers, or any other application with a MOF-compliant meta-model.



CORE© C4ISR / Systems Engineering

NetVIZ© C4ISR graphical layout

This semantic bridge will ease the creation of specialized graphical layout from engineering tools that do not natively support a C4ISR environment, or even for legacy applications that were used to capture operational requirements in the past decades.

Multi-source environment for hardware components (FPGA)

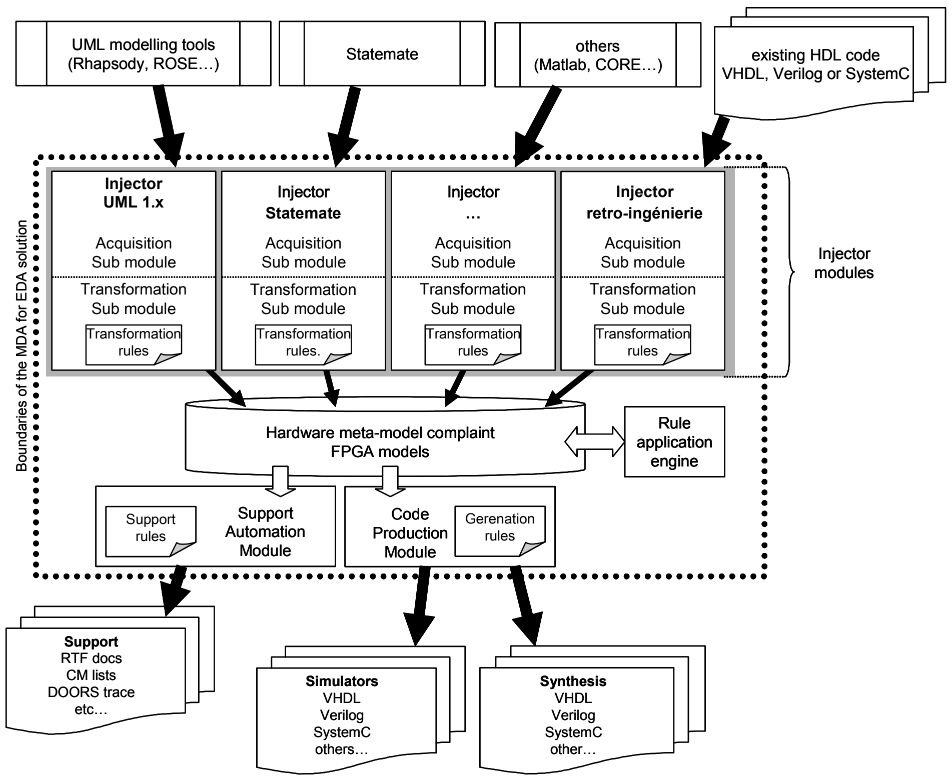
This last project illustrates how MDA can be used to link system-level engineering with EDA departments, committed to create Field Programmable Gate Arrays (FGPA) and other types of programmable logic devices.

The goal of such a project is to bring EDA people a way to reuse system-level specifications, when given as UML models. Beyond the seamless communication of the requirements, the EDA departments are facing more and more difficulties in coping with the complexity of today's FPGA: when dealing with a multi-million gate array, simulation and verification techniques used for a twenty thousand gate device may not be applicable. Then, it appears that the EDA department shall include, ahead of their existing processes and tools, supplemental activities to specify the logic of the FPGA.

Also, models of the FPGA may come from many sources like UML, Statemate, MATLAB, CORE or many others. Most of those source models can hardly be represented only with UML concepts.

This introduces the need for a hardware metamodel, that gathers information about hardware chips, ports, FSM, truth tables, etc. The generation of the VHDL / Verilog language from this hardware metamodel is made by EDA experts, while the specification on how to get this information from UML models comes from the process experts.

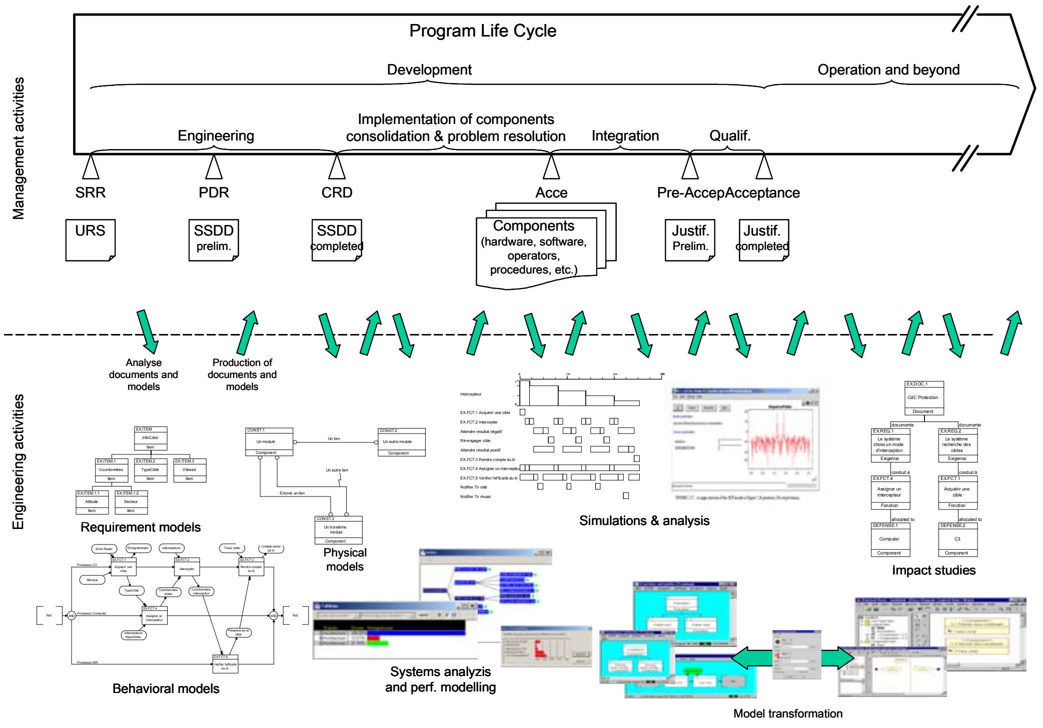
This achieves a highly flexible process, which can be tailored to the needs of different standards like ARP254 and customized to reflect the coding policies of companies, as well as their high level specification policies.


UML modelling tools (Rhapsody, ROSE…)
Statemate
others (Matlab, CORE…)
existing HDL code VHDL, Verilog or SystemC
Boundaries of the MDA for EDA solution
Injector UML 1.x
Injector Statemate
Injector …
Injector retro-ingénierie
Acquisition Sub module
Transformation Sub module
Transformation rules
Injector modules
Hardware meta-model complaint FPGA models
Rule application engine
Support rules
Support Automation Module
Code Production Module
Generation rules
Support RTF docs CM lists DOORS trace etc…
Simulators VHDL Verilog SystemC others…
Synthesis VHDL Verilog SystemC other…

This project also considers communicating with other metamodels dedicated to simulation and formal verification, achieving a real environment to design and troubleshoot high-level design before low-level design and simulations are made by the EDA engineers.

Conclusion

The MDA cannot be reduced to code generation nor to “yet another API cruncher”, it is a solution that brings innovative answers to old problems. Its main skill is to provide a model-centric approach on a life cycle basis, making modelling even more cost-effective and less dependant from tool vendors.



Statemate, UML, I-LOGIX, Rhapsody, Rational ROSE, CORE, NetVIZ, MDA, MD Workbench, Model-in-Action, Scriptor, etc. are registered trademarks of their respective owners.

Model-Driven Web Application Engineering

Pierre-Alain Muller, Philippe Studer

ESSAIM, Université de Haute-Alsace
12 rue des Frères Lumière
68093 Mulhouse Cedex
pa.muller@uha.fr, ph.studer@uha.fr

Abstract. Traditional software engineering has not been very successful in the web development area, mostly because of the gap between software design concepts and the low-level web implementation model. MDA might be a good opportunity to re-inject software engineering into web application developments, provided that the multiple web development stakeholders points of view are respected. For that purpose, we have defined a model-driven approach for web-engineering, based on a specific metamodel and supported by a graphical tool.

Keywords. Model-driven engineering, MDA, Web, Metamodel, PIM

Introduction

We found interesting to apply the MDA vision to web engineering; a field where traditional software engineering has not been very successful, mostly because of the gap between software design concepts and the low-level web implementation model[1].

We believe that model engineering and MDA, is an excellent opportunity to reconcile graphic designers with programmers, and to increase the overall productivity. The challenge is to define the right modelling concepts; to find how to introduce the power of model transformation in a world (the art of graphic design) traditionally hostile to any kind of rule enforcement.

In this paper we will first introduce the three aspects that we are modelling, then describe the new web specific modelling elements that we have defined, and finally present an overview of our metamodel for web engineering.

The work described in this paper has been done in the context of the development of Netsilon[2] a visual model-driven environment dedicated to web application development.

Modelling web applications

Aspect modelling justifies itself in the context of web modelling, because web applications are complex systems involving several different stakeholders (marketing, graphic-designers, developers, content writers…). We are working with three aspects, the business model, the hypertext model (composition and navigation) and the presentation model (see fig. 1).

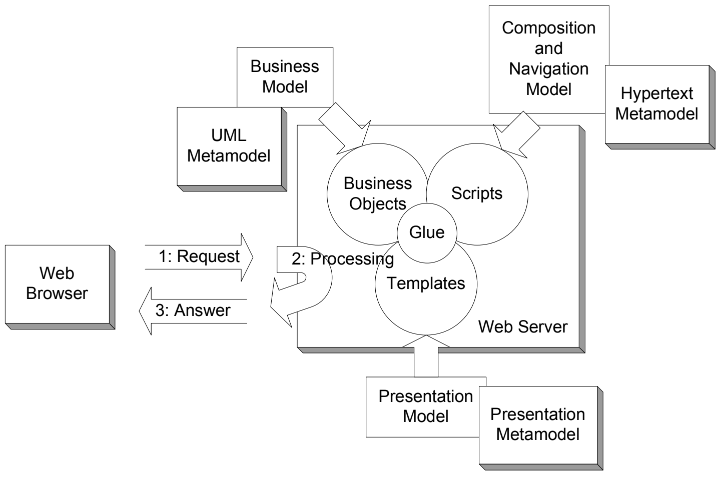


Fig. 1. Multi-aspects modeling of web applications.

The business model aspect

We use UML class diagrams to represent the business classes, their attributes and operations. We are using a kind of action language (named Xion) to describe the behaviour of the methods. In the context of UML, S. Mellor has enumerated the requirements[3] for such an action language. Xion follows a large part of these requirements, excepted for the points related to separation of data access and computation, state charts and signals. The syntax of Xion is based on the syntax of OCL augmented with Java-like control structures and affectation. Therefore, to the contrary to OCL which is a side effect free language, Xion can be used to create, update and delete instances at runtime; it has direct support for association navigation, role exploration and collection manipulation. We are currently generating either Java or PHP (and the embedded SQL statements) from the same Xion statements; in this respect, Xion is truly part of PIMs.

Besides business description, we also use this aspect to describe pervasive web services, like session management, personalization, search or statistics. When required, packages may be used to partition the business model, and separate the various concerns.

The hypertext model aspect

The second aspect, the hypertext model, is an abstract description of composition and navigation between document elements and fragments of document elements. In the context of web modelling this model describes how web pages are built and linked. In a wider context, it can also be used to handle multi-channels distribution, mixing electronic and paper documents, as we have experienced in earlier work about document modelling[4].

Composition describes the way the various web pages are composed from other pages (fragments) as well as the inclusion of information coming from the business model or current context (like sessions). Again, Xion is used as a query language to extract information from the business model and as a constraint language to express the various rules which govern page composition.

Navigation details the links between the web pages. Navigation includes the specification of the parameters that will be transferred from page to page, as well as the ability to specify actions to be applied when triggering a link. The Xion language allows the specification of the actions to be performed when transitioning from one page to another and the declaration of predicates that lead to the selection of a particular path between pages according to the current context and the business model.

The hypertext model makes it possible for a tool to check the coherence and the correctness of the navigation at model compilation time. This removes all the troubles related to parameter passing and implementation language boundary crossing (mix of interpreted languages, un-interpreted strings, parameter passing conventions…).

The presentation model aspect

The third aspect, the presentation model, contains the details of the graphic appearance of web applications.

The goal of the presentation model is to make it possible for graphic designer and HTML integrators to keep their work habits; to be able to produce static HTML pages and fragments by using conventional tools. It does not make sense to create another way of specifying the graphical appearance of web pages. Therefore, we have not provided explicit support to model the graphical appearance of the user interface, because we consider that wysiwig authoring tools are already available, and perform a good job to cover this aspect. We believe that existing tools must be integrated, without change, in the process of dynamic web page development, and that their production must be able to be controlled by an implicit presentation model.

In this vision, a dynamic web application is composed of fragments which can be developed as static HTML, supplemented with some special placeholders, easily identifiable by graphic designers and HTML integrators. Whenever some dynamic information must be inserted into a web page, the graphic designers simply designate the spot in the file where this information must be inserted.

The consequence is that we have shifted the focus of modelling to the parts that are out of the scope of these web authoring tools, and that require typically to program complex behaviour, using conventional programming languages like Java or PHP.

Modelling elements for web page composition and navigation

We have defined a small set of new modelling elements for web interface modelling. `WebFiles and Zones` which contain or reference presentation artefacts and `DecisionCenters` which explain how web files and zones relate together.

Webfiles and zones

Web files are statically associated to real files on disk (which contain data suitable for rendering in web browsers). Web files may be considered as fragments, in which case they are necessarily included in some enclosing web file (potentially also a fragment) until a non-fragment web file is reached. At the time of transformation of PIMs into executable PSMs, such top-level non-fragment web files become entry points in the web application. They are translated into target language, and are executed on the server (they can be referred to by an URL).

Zones further specialize web files to better promote separation of concerns between graphic designers and programmers. They have no statically associated files; their content is generated or retrieved at runtime, they make it possible to describe web pages at a purely conceptual level, establishing a very clean separation between logic and presentation. Using zones, software engineers can model a web user interface without entering in the presentation details, while graphic designers can focus on appearance, using conventional tools as if they would be doing static web sites.

Decision centers

Decision centers represent variation points in the web user interface. Each of these centers is responsible for a decision to be taken at runtime, while the user is interacting with the system. Decision centers support an entry action which has access to session variables and may define local variables.

We have identified six kinds of decision centers to cover the complete range of variation points in web user interfaces.

Detailed metamodel for web page composition and navigation

Figure 2 presents an overview of our metamodel for web page composition and navigation. The central element is `WebFile` which describes a document element (or a fragment). A web file can be treated as a static or a dynamic element: if static, it is simply copied unmodified during deployment; if dynamic, it is generated as server side code and participates in the dynamic part of the web application. HTML tag filtering and striping makes it possible to use web authoring tools for the design of fragments as easily as for the design of entire pages.

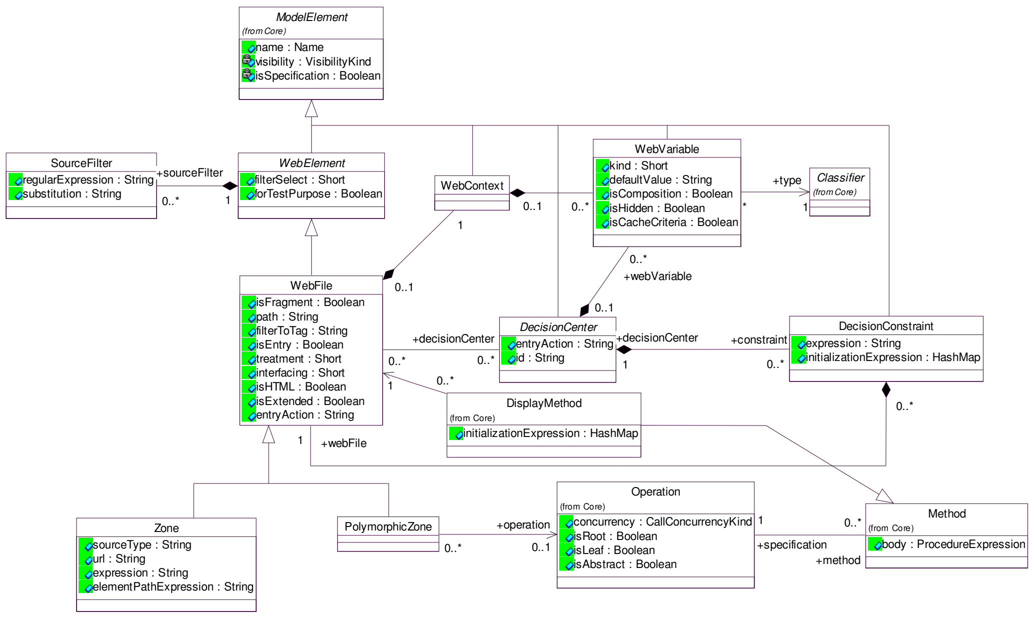


Fig. 2. Excerpt of the hypertext metamodel for web page composition and navigation.

A web file can be specialized as a `Zone` whose associated content is obtained dynamically by the evaluation of an expression that returns the URL of the content or the content itself.

A web file can also be specialized as a `PolymorphicZone`. A polymorphic zone is the mean to introduce the object notion of polymorphism in the hypertext model. In fact, a polymorphic zone is associated to an `Operation` that is in charge of producing the content. Since the operation can be implemented by overridden methods, the content can be generated according to the real class of an instance. To reinforce the separation between the business model and the hypertext model, we introduce a subclass of `Method` named `DisplayMethod` that is associated to a web file. The production of content by an `Operation` implemented by one or more display methods is thus realized by webfiles.

Each web file has a context `WebContext` that describes its entry parameters. A parameter is described by `WebVariable` and has a type which is an instance of `Classifier`.

`DecisionCenter` define variation points in the hypertext model. A decision center has an *entryAction*, a unique *id* to identify its placeholder, local variables (`WebVariable`) and an ordered sequence of `DecisionConstraint`. A decision constraint defines a guard whose evaluation to true leads to the selection of its associated web file.

Transformation of PIMs into PSMs

Several different deployment platform can be defined for the same project. A deployment platform specifies a target environment and a persistence system. The encapsulation provided by the decision centers allows the design of effective PIM for the hypertext model. The business model in the PIM is transformed as a set of classes in the target language and a database schema. The hypertext model is transformed as a set of static documents, classes or scripts in the target language. Finally, the action language is translated in the target language.

A deployment platform specifies a target environment (web application server, language, site information). Our code generator currently targets any combination of (PHP, J2EE) applications server with (Oracle, MySQL, PostgreSQL) databases.

Conclusion and future work

We have applied model-driven engineering in the field of web engineering. We believe that the MDA initiative is a good opportunity to raise the level of abstraction, and increase the productivity in the development process of web applications, provided that the overall process does not change too much the working habits of graphic designers and HTML integrators. Model developments remain the duty of software developers.

We have favoured the development of a specific metamodel dedicated to web application development (instead of a profile) to better cope with specialized behaviour and tool user-interface. This metamodel has to be used in conjunction with UML and promotes concerns' separation between graphic development and software development.

Our experience shows that web user interfaces can be modelled using a small and well defined set of modelling elements (3 types of web files and 6 types of decision centers). Our metamodel has been validated by the development of about a dozen of significant totally model-driven web applications (for online examples visit http://www.domaine.fr, http://www.namestep.fr/, http://www.solinest.fr/, http://www.mydocmail.net, http://www.prh-france.fr). We can generate various executable PSMs from the same PIMs.

In the future, it would be interesting to re-align our action language with the on-going standardization efforts led by the OMG; this involves better understanding the interactions between OCL, the action semantics and QVT. Another major point would be to make the PIM to PSM transformations explicit, and therefore allow customization of the code generation phase.

References

[1] H.-W. Gellersen, M. Gaedke, "Object-Oriented Web Application Development", IEEE Internet Computing, pp.60-68, Januray-Frebruary 1999.
[2] W. El Kaim, O. Burgard, P.-A. Muller, MDA Compliant Product Line Methodology, Technology and Tool for Automatic Generation and Deployment of Web Information Systems, Journées du Génie Logiciel, Paris, Décembre 2001.
[3] S. Mellor, S. Tockey, R. Arthaud, P. Leblanc, "An Action Language for UML: Proposal for a Precise Execution Semantics", UML 98, LNCS1618, pp. 307-318, 1998.
[4] 5. M.-C. Roch, P.-A. Muller, B. Thirion, "Improved flexibility of a document production line through object-oriented remodeling", Second congress IMACS, Computational Engineering in Systems Applications, Hammamet, CESA'98, Vabeul-Hammamet, Tunisie, Vol III, pp 152-159, april 98.

MDA at LIP6

The LIP6 (Laboratoire d'Informatique de Paris 6 = Computer Science Lab. of Pierre et Marie Curie University, Paris, also known as University Paris 6) was created on January 1st, 1997, as a merger of the previous three computer science laboratories of the same University (LAFORIA, LITP and MASI).
With 450 collaborators, it is one of the most important computer science laboratories in France. LIP6 is structured in 9 teams with research domains ranging from hardware to Artificial Intelligence.

Work on MDA at LIP6

Two teams work together on MDA, namely SRC (Systèmes Répartis et Coopératifs = Cooperative and Distributed Systems) and OASIS (Objects and Agents for Simulation and Information Systems). The total number of people directly involved with MDA is about 7.

SRC wants to make use of MDA for building distributed systems (which is its primary aim) and conversely adapts techniques from distributed systems to the construction of MDA tools chains (e.g. an Open Source platform for MDA called *ModFact*, which provides an implementation of MOF, XMI and JMI standards).

OASIS is interested in the representation of the various kinds of knowledge involved in software development. Some parts of this knowledge appear in the form of models and meta-models, other parts as transformation rules (e.g. PIM to PSM). Work on the *MetaGen* development platform (that supports all the MDA concepts), as well as experiments in its use, started as early as 1990. Right now, the OASIS group is engaged in building (with MetaGen) a prototype CASE tool for JEE based applications as a part of the French IMPACT project.

Together SRC and OASIS have structured their research on MDA along three directions:

- Model Transformation
- Model Integration
- Model Methodology

Model Transformation

In our opinion, Model Transformation is a key issue of MDA, on which we have accumulated a substantial amount of experience through the use of the MetaGen tool. Developing an application with MetaGen involves two types of model transformations, one which is analogous with PIM to PSM, and one for code generation. In our practice, such transforms are effected by rulebases (along the lines of Expert Systems) operated by a first-order, forward-chaining and object-based inference engine (called Neopus).

Right now, we intend to represent transformations as models, and to evaluate these models using a generic and dynamic engine. The idea is to define an engine that can dynamically performs all kinds of transformation. The goal is to define the architecture for such an engine and to provide an open source implementation in the ModFact platform.

The OMG has defined an RFP for model transformation named MOF 2.0 Q/V/T. Its goal is to define a language, in the M2 space, for model transformations. LIP6 as an OMG member is involved in the RFP submission process and has submitted an LOI for such a standard. It belongs to the OpenQVT submission group. The repository of transforms that were developed and tested with MetaGen provides a good benchmark for Q/V/T.

Model Integration

The MDA approach requires lots of different tools such as modelers (meta-model compliant editors) for building models, repositories for storing models, engines for transforming models, and others tools such as for example reverse engineering ones or simply documentation generators. Note however that most model-manipulating tools share the same basic structure as transformation tools, which should ease interoperation.

There is an important need of interoperability between all these tools. The OMG has defined the XMI format, that can be used to resolve partially some problems of interoperability but it is not enough. XMI does not deal with transport: you can send XMI documents by e-mail, by file transfer, etc.

As a builder of MDA platforms (ModFact and MetaGen) and because it has developed bridges between them, the LIP6 is aware of the tool interoperability problem. Moreover, its skills in middleware are very appreciated for this issue. Right now, the LIP6 studies how the Web Services standards together with XMI could guarantee interoperability between tools. The idea is to consider each tool as a provider of services. The particularity of these services is that they are model-based since the information that they input (or output) are models. Thus it should be possible to adapt Web Services standards to the model world using XMI. We are defining the architecture of a model bus and we plan to implement it as an open source tool.

Model Methodology

Basically, MDA deals with PIM and PSM and the transformation from PIM to PSM. But this can be very complex. For example, several PIMs can be needed to build PSMs. In another way, PIMs may need to be transformed in other PIMs in order to be transformed into PSMs. On the other hand, models of platforms are needed (Platform Description Model). MDA should encompass all these aspects in order to help users. To this end, a methodology should be provided in association with MDA**.** The substantial experience accumulated with MetaGen will be used to provide such a methodology by means of knowledge elicitation techniques.

The LIP6 has a lot of experience in methodology. First, it applied the concepts of the MDA approach since 1990 on industrial projects. Second, it has participated to the elaboration of standards such as RM-ODP.

Right now, the LIP6 thinks about structuring the PIM to PSM transformations into several well-defined layers. These layers can be seen as phases of a development process. All information related to each of these layers will be encapsulated into methodological components. The idea is to provide off the shelf methodological components. For example, a methodological component could be “conception with CORBA for Real Time”. We aim at defining such methodological components and to provide recommendation for using them in order to build global PIM to PSM transformations.

References

Projects:

RNTL TRAMs
The goal of the TRAMs project is to provide a strategy for migrating information system to the web and to provide tools that implement this strategy.
http://www.industrie.gouv.fr/rntl/AAP2001/Fiches_Resume/TRAMS.htm

IST COACH:
COACH is concerned with the rapid and cost effective development of large scale and mission critical distributed applications by using software components.
http://coach.objectweb.org

RNTL IMPACT
The goal of the IMPACT project to develop in Java software components for the ObjectWeb platform. These components must conform to the specifications issued by JCP or by OMG.
http://www.industrie.gouv.fr/rntl/AAP2001/Fiches_Resume/IMPACT.htm

Tools:

MODFACT:
ModFact is an Open Source project for tooling the MDA. It provides an implementation of MOF, XMI and JMI standards.
http://modfact.lip6.fr

METAGEN:
MetaGen is not currently available. An Open Source distribution is being finalized.

Publications:

G.Blain, Huet B., Lesueur B : *Medical record-oriented meta-modeling : the adjustment process*, in XIXth International Conference on Medical Informatics in Europe 2003

M.P. Gervais, *Towards an MDA-Oriented Methodology*, in Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02), IEEE (Ed), Oxford, England, August 2002, pp265-270

N. Revault : *Model transformation based on production rules*, SET'02 workshop at ICGT 2002, Barcelona

N. Revault, H.A. Sahraoui, G. Blain & J.-F. Perrot : *A Metamodeling technique: The MétaGen system* Tools 16: Tools Europe'95, pp. 127-139,

MDA experience and deployment at TOOL OBJECT

Florence BASCANS	TOOL OBJECT	C.E.O.	fbascans@toolobject.fr
Josiane COLIRE	TOOL OBJECT	R&D director	jcolire@toolobject.fr
Martial CHRISMENT	TOOL OBJECT	ECM Architecture manager	mchrisment@toolobject.fr

TOOL OBJECT is an IT company providing an Enterprise Component Model (ECM) for the banking market based on a Model-Driven Architecture (MDA).

Business knowledge is capitalized and organized within this model, making analysis and conception more powerful and development more efficient on any technical framework.

Background

In 1997, the creators of TOOL OBJECT drew the following statement from their 10 years of experience in the banking domain : « too many IT projects have been built by defining the technical architecture before even thinking about the business architecture ».

The problem with such an approach is that business depends strongly on technique. The result is that solutions developped are fragile and, when facing a constantly changing technology market, they soon become obsolete. From the start, TOOL OBJECT wanted to oppose this tendency by creating business minded solutions rather than technical minded solutions.

Thus, the ECM has naturally evolved towards an MDA Architecture which was the only one to guarantee full independence from the technical side.

TOOL OBJECT experience

To succeed in this challenge, TOOL OBJECT relied on market standards in conception, mainly UML and Rational Rose. It also centered its offer around the MDA approach and its different levels of abstraction :

Increasing level of abstraction ↑

Information Type	*Example*
Meta-metamodel	MOF model
Metamodel	UML
Model	ECM
Data	Banking data

The Entreprise Component Model (ECM) created by TOOL OBJECT is a banking Platform-Independent Model (PIM). As well as gaining in flexibility during the development phase, this approach has the main advantage of making the models easily understandable to people with no technical knowledge.

Using MDA, the ECM is an agile solution which offers comprehensible models with the ability to incrementallaty change with the organization of the banking business around business domains (credit, savings, means of payment, etc…).

To efficiently transform a business model (PIM) into a technical one (PSM), TOOL OBJECT quickly realized that it was necessary to think in terms of repetitive technical issues (persistance, distributed processes, workflow management, …) rather than in terms of platform-specific technical solutions (EJB, Servlet, Java Bean, …).

To achieve this, new stereotypes and new properties have been added to the business concepts in order to take into account generic technical issues such as :

- For persistance, each class and attribute in the ECM references the name of the relational table or column in which the data is stored,
- For distributed processes, the ECM uses Rational Rose stereotypes called « api » on the class operations in order to identify the business services offered by each concept.

The ECM thus provides all the elements to be automatically derived into a PSM for any programming language. For each technical issue, the programming language often offers an appropriate technical solution (Ex : EJB Entity Bean is a solution for persistance management, EJB Session Bean for service distribution, …).

The job of transforming the ECM (PIM) into a PSM mainly consists in defining and then automizing generation rules for the technical concept.

TOOL OBJECT has developped, since 1998, a full banking solution written in C for a bank in the Toulouse region of France. For a year, TOOL OBJECT has been successfully working on the definition of a new solution in Java based on its ECM. Its approach has proven to be highly flexible, quickly deriving the business model from one technical platform to another without any major changes.

<u>Development Process</u>

The development process of the ECM solution is organised around the transformation of the ECM into a platform specific model for the client :

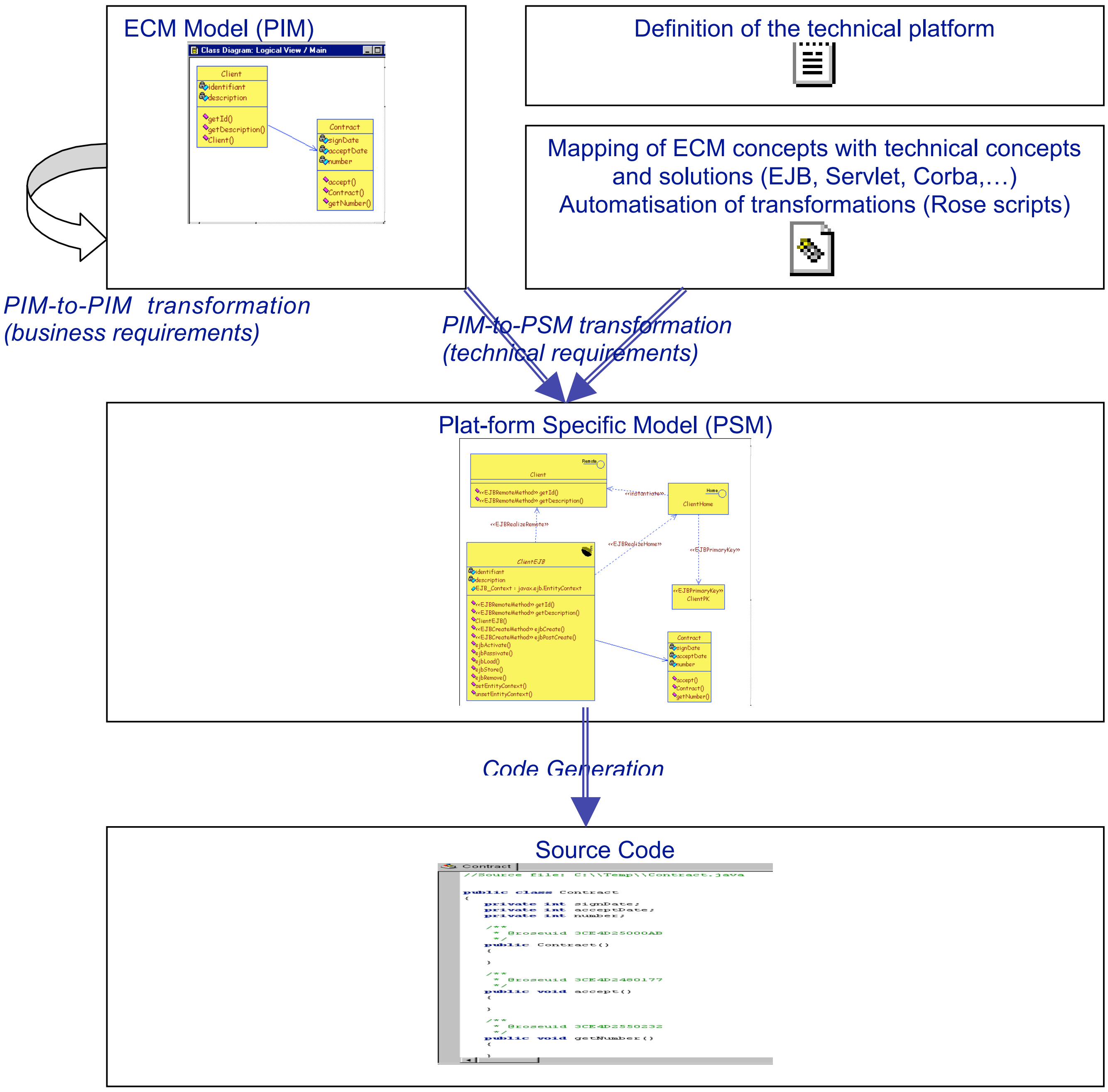
- Adapt the ECM to the client’s modelling rules : PIM-to-PIM transformations,

- Personnalize the ECM to fulfill the client’s business requirements either manually or automatically (e.g by merging the ECM with a PIM model of the client).

- Map the ECM business concepts (PIM) and the technical concepts or solutions specific to the client platform,

- Apply PIM-to-PSM transformations using Rose Scripts or technical patterns (Ex : pattern EJB, Servlet, …)

- Code generation from the PSMs.



```
//Source file: C:\\Temp\\Contract.java

public class Contract
{
    private int signDate;
    private int acceptDate;
    private int number;

    /**
     * @roseuid 3CE4D25000AB
     */
    public Contract()
    {

    }

    /**
     * @roseuid 3CE4D2480177
     */
    public void accept()
    {

    }

    /**
     * @roseuid 3CE4D2550232
     */
    public void getNumber()
    {

    }
```

Through the use of MDA transformations, the ECM becomes the unique reference from which all new models (PIM and PSM) are derived according to business requirements and technical constraints of the client. TOOL OBJECT has thus achieved its goal by creating a business-minded rather than technical solution.

Conclusion

To fullfil its challenge and integrate the MDA philosophy, it is necessary to operate a revolution both cultural and organisational within the teams by separating clearly the business aspects from the technical or development ones but the result is worthy :

- Business knowledge is no longer polluted by technical issues facilitating the transfer of expertise between people,

- the solution is stable and evolutive with a strong adaptability to new technologies,
- risks are more controlled, the technical problems having no impact on the PIMs,
- stability and performance can be optimized by choosing the best technical platform available to deploy the PIMs.

The MDA vision at INRIA

Jean Bézivin and Patrick Valduriez, Atlas Group, INRIA and IRIN, Nantes
Jean-Marc Jézéquel, Triskell Group, IRISA, Rennes
Raphael Marvie and Jean-Marc Geib, INRIA & LIFL, Lille

Among several teams that are investigating on model engineering in systems and software at INRIA, three research groups are specifically involved in MDA approaches. Triskell the first one is located in Rennes, Atlas the second one is located in Nantes and Jacquard the third one is located in Lille. They have strong links together and also with the group of prof. M. Bouzhegoub at Versailles described elsewhere in this report.

The Triskell group
(http://www.inria.fr/recherche/equipes/triskell.en.html)

The research domain of the Triskell project is the reliable and efficient design of software product lines by assembling software components described with the UML. Triskell is particularly interested in reactive and distributed systems with quality of service constraints.

Triskell's main objective is to develop model-based methods and tools to help the software designer to obtain a certain degree of confidence in the reliability of component assemblies that may include third-party components. This involves, in particular, investigating modeling languages allowing specification of both functional and non-functional aspects and which are to be deployed on distributed systems. It also involves building a continuum of tools that make use of these specification elements, from off-line verifiers, to test environments and on-line monitors supervising the behavior of the components in a distributed application.

Another goal of the Triskell project is to explicitly connect research results to industrial problems through technology transfer actions. This implies, in particular, taking into account the industrial standards of the field, namely UML, Corba Component Model (ccm), Com+/.Net and Enterprise JavaBeans.

Triskell is at the frontier of two fields of software: the field of specification and formal proof of software, and that of design which, though informal, is organized around best practices (e.g.; Design Patterns or the use of off-the-shelf components). We believe that the use of the techniques that we develop will make it possible to improve the transition between these two worlds, and will contribute to the fluidity of the processes of design, implementation and testing of software. Three main research directions are explored within the Triskell project.

Contract-based design

One of our objectives was to strengthen the contract based approach for OO designs. In that direction, a new model for contract aware components (based on a 4 level decomposition) has been defined in, and now serves as a reference in this domain (for instance it has been adopted by the SEI). In relationship with the European QCCS IST project ("Quality Controlled Component-based Software"), UML extensions for specifying quality of service properties of UML2 components have been defined. This approach has been presented as a tutorial called ``Model-Driven Engineering with Contracts, Patterns, and Aspects" at AOSD 2003 (2nd International Conference on Aspect-Oriented Software Development).

Meta-modeling and model-transformation

Triskell has obtained many results in the fields of meta-modeling and model transformations, that have been prototyped with the UMLAUT tool (http://www.irisa.fr/UMLAUT). A weaving mechanism within the QCCS weaver and modeler platform, to weave QoS aspects (and especially real-time ones), is now available. Meta-modeling is also addressed, since we have defined ways of expressing design patterns, aspect-oriented mechanisms and more widely frameworks within the UML meta-model. One of the consequences of these definitions is the enhancement of the current work on model transformations to guarantee, during refinement and deployment stages, a stated QoS or any other non functional properties: a first result was obtained with testability. The integration of formal methods (MSCs scenarios, models for true concurrency) to the UML practice, in terms of model transformation has been studied. A model for true concurrency has been properly defined in and is under connection to the Umlaut framework: test generation techniques should be available for deployed systems with deployed embedded testers. Moreover, we have defined how to obtain MSC projections that allow the abstraction of complex interactions. This the first step, in terms of model manipulation and transformation, to allow the creation of behavioral assets (especially in a product-line context).

Model-Based Testing for components and component-based systems

Triskell has done some pionnering work on the notion of UML model based testing, with the very first results published at UML'98. Some of the scientific hard-points have been overcame and a prototype connection of off-the-shelf test generation tool to the Umlaut framework is effective and validated on real-world case studies (Gemplus, FT R&D). Still, more work is needed both at the conceptual level (e.g.; how to efficiently encode sates made of object graphes), as well as the implementation level to make this prototype usable by third parties. A dedicated test language to specify high level test objectives ---and represent the generated test cases--- using the UML sequence diagrams has been developed: it forms the basis of the UML test profile currently under standardization at OMG [25]. The assembling of components has also been studied and efficient algorithms have been defined and integrated to the Objecteering Case tool. They automate the ordering of integration by minimizing the testing effort (by reducing the creation of stubs during components integration).

The Triskell research agenda

Within the general context of model driven engineering for component based soft real-time systems, Triskell identifies four complementary research directions: analysis of models, design processes based on contracts, patterns and aspect weaving, model based testing, and meta-modeling tool support. While the static dimension of modeling is quite well understood, the major challenge in all these directions is to properly deal with the dynamic dimension of modeling, including its proper integration with static concepts such as encapsulation, inheritance and polymorphism.

The Atlas Group
(http://www.inria.fr/recherche/equipes/atlas.en.html)

Today's hard problems in data management go well beyond the traditional context of Database Management Systems (DBMS). These problems stem from significant evolutions of data, systems and applications. First, data have become much richer and more complex in terms of formats (e.g. multimedia objects), structures (e.g. semi-structured documents), content (e.g. incomplete or imprecise data), size (e.g. very large volumes), and associated semantics (e.g. metadata, code). The management of such data makes it hard to develop data-intensive applications and creates hard performance problems. Second, data management systems need to scale up to support large-distributed systems (such as the Internet or cluster systems) and deal with both fixed and mobile clients. In a highly distributed context, data sources are typically in high numbers, autonomous and heterogeneous, thereby making data integration difficult. Third, this combined evolution of data and systems gives rise to new, typically complex, applications with ubiquitous, on-line data access: virtual libraries, virtual stores, global catalogs, services for personal content management, services for mobile data management, etc.

The general problem addressed in Atlas is *complex data management in distributed systems*. The objective is to design and validate new solutions for this problem with significant advantages in functionality and performance. To tackle this objective, we separate the problem along three main dimensions. The theme "models and summaries" addresses the issues of data abstraction from complexity and size. The theme "multimedia data management techniques" deals with efficient and personalized access to multimedia data. Finally, the theme "distributed data management techniques" addresses the problems of data replication and distributed query processing with complex data.

DBMS has become a very mature technology that is ubiquitous in information systems. Over time, the extensive use of DBMS technology has had major consequences in large organizations: the production of very large databases, the production of heterogeneous databases, and the increasing requirement of diverse applications to access those very large, heterogeneous databases. This creates difficult technical problems, which get worse as DBMS technology improves and is more able to produce very large, heterogeneous databases. We believe a common answer to these problems must rest on a generalization of the principle of data independence, applied to instances (e.g. relational tuples) for very large databases as well as to data descriptions (e.g. schemas) for heterogeneous databases and applications. Therefore, we pursue two independent research actions, one on summary management and another on model management, and a third federating action on the integration of heterogeneous summaries and models.

The team working on model management has a previous experience with meta-modelling frameworks similar to the CDIF standard. More than ten years ago they proposed a system called sNets, a typed partitioned and reflective version of semantic networks. They build a first prototype in Smalltalk on the work of the PIE system, a proposal made in the early 80's by Bobrow and Goldstein at PARC. This prototype was transferred to the Soft-Maint group and the underlying ideas are still used today in the kernel of the Semantor™ reverse engineering tool or MIA-Studio ™ , a meta-model based generic transformation system. The sNets system as built on top of a minimal meta-meta-model and used a collection of domain dependent meta-models. When the MOF and UML formalisms were proposed at OMG, it was clear that we had a similar

4-level framework. We are still today experimenting with this platform, re-implemented today in Java. The main difference between this sNets research platform and the MOF industrial standard is that one of our main design goal has been minimality of concepts. The major lesson learned in this work is that minimality of the conceptual implementation does not mean reduction in functionality.

The ATLAS group is deeply involved in research on model engineering. We firmly believe that a reasonable level of platform independence could be achieved today in software engineering like it was achieved in the seventies by the Database community when the relational model was proposed and the SQL language was later accepted. The rise in abstraction may come today from the MDA approach and formalism like UML and MOF may play a role similar to SQL in the DBMS field.

Above all we do believe that model engineering will have an important impact on many fields of computer science like system engineering, software engineering and data engineering among others. The rise in abstraction will come from the systematic application of a number of unifying principles. The most important of these is to consider "**models as fist class citizens**", alike objects were considered in the early 80's. This will probably bring us much beyond object and component technology if we take seriously the second principle stating that "**transformations are models**". This second principle is one of the assumptions of the recent RFP by OMG, namely the MOF/QVT RFP.

We are presently building a framework called ATL (for ATLAS Transformation Language), a declarative language and associated framework that will hopefully be aligned with the resulting specification of the MOF/QVT RFP when it will converge. The intention is to provide the transformation engine as a free software component at that time.

More than just a transformation engine, we are interested in a third, even more demanding, principle stating that "**models are assets**" (i.e. they are composable and specializable) and its direct corollary that "**transformations are also assets**". This means that a company may buy a library of transformations (for example from UML 1.5 to J2EE 1.4) and later adapt it to suit its own needs. Transformations may thus be composable, adaptable and their properties should be explicitly verifiable. At this condition, we are on the verge of witnessing the apparition of a market of rich structured libraries of transformations, similar to the libraries of several hundred of classes that appeared in Smalltalk-like systems at the beginning of the object-oriented period. The notion of MDA component is rapidly taking shape. The ATLAS group is really interested in experimenting these exciting new possibilities, mainly in the domain of software maintenance and data migration.

The Jacquard group

(http://www.inria.fr/recherche/equipes/jacquard.en.html)

1 Context

The context of our work around the Model Driven Architecture is to study the feasibility of defining abstractions for functional and non-functional aspects, as well as execution models, in order to produce extensible and adaptive distributed applications. In particular, our interest is to apply the separation of concerns from meta-models to applications, as well as to study the weaving of aspects / concerns using model transformation.

1.1 MDA for Software Architectures

Our first experiments regarding the MDA were related to software architectures. Jointly using meta-modeling and separation of concerns we proposed in a scheme to structure meta-models of software architectures. The aim of this work is to support the handling of software architectures regarding the architectural concerns of software engineering processes, thus to offer to each actor of such processes a dedicated view of a software architecture.

In addition to this organization of meta-models for software architectures, this work demonstrates the feasibility of generating dedicated environments for handling software architectures using the defined separation of architectural concerns. Thus, actors of software engineering processes are provided with dedicated tools to address their concerns and to ease their cooperation around the architecture they are defining.

1.2 MDA for Middleware

The second motivation of our work is to define and provide abstractions of middleware in order to allow their use at design time of distributed systems. These abstractions are intended to cover non-functional aspects provided by middleware—like transactions and persistency—as well as execution models—like RPC-based or model-oriented middleware.

Providing these abstraction should allow designer to express their system requirements in an abstract fashion, thus co-designing applications and middleware. Based on this information, and on the use of model transformation, we are expecting to produce dedicated and adaptive middleware support for distributed applications. In the meantime, another major aspect of our current work is related to middleware implementations in order to master concepts of middleware as well as their implementations.

2 Working with the MDA

2.1 Models Everywhere Every time

Reifying models To be used a model definition has to be reified. Nevertheless, most proposals tend to use model reification only to perform computations on a model—like mapping a PIM to a PSM—meaning that models are not easily available otherwise. Thus, using model driven engineering is foreseen as being a quite static approach. Deployment and reconfiguration are example of activities that could take benefit from full-time access to system models. Having access to a system definition could only improve the capabilities of an environment to support adaptability of applications.

Implementation Our approach regarding model availability is to use model repositories as part of environments—from application design-time to runtime—in order to ease model use. At design time, such repositories improve the collaboration of the various actors of a software engineering process. At runtime, this information still describes the application and help supporting reconfiguration of applications.

Thus, two levels are co-existing in this context: a component request broker for parts of the applications to communicate together, as well as a model request broker for access and handling of models. Beyond COTS components, this approach introduces COTS models that should be reusable to improve existing applications or to produce new ones.

2.2 Processing Models

Our intent is to support separation of concerns while providing seamless integration of functional and non-functional aspects of systems.

Aspect Oriented Modeling Modeling software architectures results in using Platform Independent Models as architecture descriptions. Using refinements is it then possible to incrementally introduce aspects in architecture definitions. To do so, the refinement process relies on transformation aimed at composing architecture models with aspect models. Thus, adaptation of a component model is also defined as a model providing flexibility, which coupled with contracts, allows the introduction of semantic to transformations. This PIM refinement is then mapped to Platform Specific Models for the JAC (Java Aspect Components) and OpenCCM platforms.

Views Oriented Design Studying the design of reusable components, and the expectation of their functional adaptability, lead some of us to study the use of functional views in the context of the MDA. The approach is to begin with views oriented design to make views reusable in various contexts—e.g. various information systems. This results in the definition of adaptable component models, as well as sets of associated rules for design and assembling. Moreover, theses models can be mapped to platforms such as EJB or CCM.

2.3 Supporting the MDA

In order to support a MDA process, tools have to be provided. In the meantime, these tools are applications. Thus, the MDA should apply to their definition—even if a bootstrap is required for the first steps, it should be completely reproducible using an MDA process.

Our approach on that point is to define MDA components to be used in order to assemble an MDA production line. To do so, it is important to offer model repositories as components. Thus, an MDA production line could easily be connected to models representations. In the meantime, an MDA process is a set of transformations—refinements, mapping, and integration of models. Thus, transformations also have to be available as components. In addition to these MDA components, it is required to provide a means for their assembling. This means is comparable to software architecture description, which in this context is defined as a model. We are now working on the implementation of a transformations engine as well as a framework to define transformation components. Being model driven, the intention is to define MDA production lines using models, and to generate most of their implementation applying the MDA principles. These experiments are right now performed on top of our OpenCCM platform.

3 Collaborations

3.1 LIFL, MDA, and Standards

Since 1998, the LIFL has been involved in the transfer of research results through platforms as well as through standardization. Our first contribution was the elaboration of the CORBA Scripting language *Idlscript*. Since 1999, we have joined the process of standardizing the CORBA Component Model for which we were responsible of the Finalization Task Force.

We have also contributed to the definition of the UML Profile for CCM specification. Currently, in addition to being responsible for the CCM Revision Task Force, we are involved in the MOF 2.0 Query/View/Transformations as well as the MOF 2.0 to IDL 3 mapping specifications.

3.2 Labs and Projects

We currently have collaborations with various research labs including the LIP6 (Paris, France) and the DSTC (Brisbane, Australia). Our work with the LIP6 aims at providing a platform for building MDA production lines (ModFact project). Our collaboration with DSTC is related to model transformation and aims at discussing a joint submission for MOF 2.0 Q/V/T. Finally, the LIFL also takes part together with Thalès, CEA, and other INRIA Research teams in the Carroll research project—which aims at studying the use of model driven engineering for constraint systems

MDA: Some Lessons Learned from Data Base Technology and Design
(Extended abstract)

Mokrane Bouzeghoub

INRIA and Laboratoire PRiSM, Université de Versailles, France
Mokrane.Bouzeghoub@prism.uvsq.fr

1. Introduction

The Model Driven Architecture (MDA) recently proposed for discussion by OMG put a strong emphasis on fundamental notions such as models and meta models, mappings and derivation, code generation and application deployment, components and software architectures, analysis and design methodologies, evolution and maintenance, etc. Most of these terms have already been used in software engineering and database technology, but the MDA considers all of them under a uniform approach of *modeling* and *mapping*. Apparently, at first glance, the MDA approach is not different of what the Database community has been doing during several decades, applying it to a restricted area of data management. This paper is a very short summary of the main features that characterize the Database domain, with a particular focus on those concepts, which have a strong link with the MDA philosophy, in the perspective of their reuse within this new context.

The Database domain can be studied following two complementary tracks: the core technology track and the design track. We focus on system architectures and design methodologies, which might have a link with the MDA approach.

2. The Database Core Technology

The Database core technology is based on some strong features which were considered as corner stones for database design and evolution. Among these features, let us mention the following:

- The first feature is *Logical-physical independency*, which means that a process defined at the logical level is not biased by data structure changes happening at the physical level. Database queries are defined over the logical schema and programmers are not aware of the physical organization of data. To increase the effectiveness of this feature, the View mechanism allows to define queries over views and consequently to make the user queries themselves independent of the logical schema, hence facilitating its evolution without having to rewrite and recompile thousands of user queries. There might be a hierarchy of views (views defined over other views). If all users interact through views, adding a concept at the logical level does not cause any change at the query level, unless users want to mirror in their own views the evolution of the logical schema. However, removing a concept from the logical schema may lead to some undefined views (those defined over these concepts).

- The second feature is *interoperability*, that is the ability given to applications to operate on autonomous, distributed and heterogeneous databases. This architectural feature opens doors to multiple systems integration and to distinct enterprises interaction. Interoperability can be implemented by means of a wrapper-mediator architecture or by using a software bus such as Corba. The wrapper-mediator architecture is based on the use of a global schema (e.g. integrated schema) while the software bus is based on the use of a common language (e.g. IDL in Corba).

- The third feature is the *control through meta data*, that is any real world artifact (whether it represents a data element, an integrity constraint, a privacy constraint, a business rule, a transaction, etc), which should be known by or is under the control of the database system, is represented by metadata. Metadata management is one of the base functions of any database system.

These fundamental features have been implemented in various database system architectures, ranging from a centralized architecture to a client-server architecture, a three-tiers architecture or a mediation architecture. Mediation architectures allow a uniform access to a federation of databases or to a set of distributed and heterogeneous data sources. In the mediation architecture, the mediator is based on a global schema which can be seen as an integration of user requirements on one hand and as a mirror of what distributed sources can provide in terms of data on the other hand. User views are defined by their respective schemas and by the queries that compute them from the mediation schema. User queries are expressed on user views. The operational link between the mediation schema and the source schemas is defined by mediation queries. Within this architecture, we generally distinguish between three kinds of schemas:
- A specific database schema, called local schema or data source schema,
- An integrated schema describing the whole of local sources, called global schema, virtual schema or mediation schema,
- A client schema describing a specific user view, also called view schema or external schema.

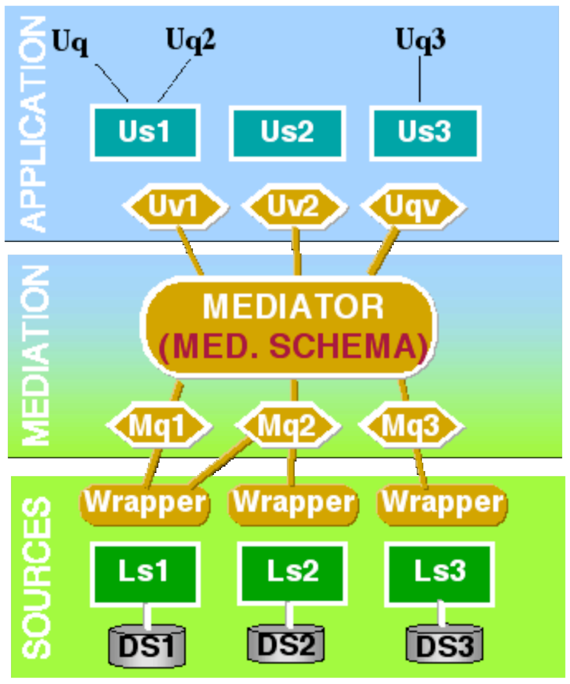


Figure 1: Mediation architecture

One of the major issues in these architectures is to define the mappings between the different schemas. Two orthogonal types of mappings exist within a database system: mappings between logical and physical schemas in one hand, and mappings between client views and the global schema or between the global schema and data source schemas in other hand. For an illustration purpose, we consider these latter ones.

Mapping between client level and mediation level

Each user query *Uq* defined over a given user view *Uv* is then mapped onto another query *Uq'* defined over the mediation schema by substituting in *Uq* each virtual table name by its definition as a query over the mediation schema (Uq' =S(Uq/Uv)). This is a very well known rewriting process in all database systems.

Mapping between mediation level and source level

This mapping is approached by two different rewriting processes: the "global as view" process (GAV) and the "local as view" process (LAV).

- The GAV approach defines each entity type of the mediation schema as a query over source schemas. The rewriting process in this case is almost similar to the previous one. Each user view *Uv* (or query) defined over the mediation schema is rewritten by substituting mediation table names by their query definition over the sources. This approach is known for being easy to implement (similar to the view mechanism) but also for its non flexibility: a change at the source level may lead to the change of many and maybe all of the mediation queries.
- The LAV approach defines each entity type of each source schema as a query over the mediation schema. The rewriting process is a little more complex but it constitutes an interesting mapping which has the advantage of flexibility and scalability: a change at the source level results into a change of one mediation query only, and adding a new data source consists in adding only queries that define this data source.

3. The Database Design

Following and complementing the Database core technology, the Database design process is concerned by all activities which help in the design, validation and implementation of databases. A lot of effort was put on the design of relational databases, thanks to the formal basis provided by the relational model and to the widespread dissemination of relational database systems. The database design process is based on some fundamental features which drive the analysis, conception and validation of a given database application. Among these features, let us mention the following:

- The first feature is the use of a hierarchy of *abstraction levels*. Indeed, we have already seen that Database technology imposes two abstraction levels: a *logical level* and a *physical level.* This hierarchy is complemented by a *conceptual level* whose goal is to capture as much semantics as possible from the real world, independently of any specific underlying database technology. Consequently, the roles of the three abstraction levels can be summarized as follows: (i) The conceptual level aims at representing the semantics of a given real world without any concern for database technology, technical constraint nor performance. Its main goal is only understanding and describing in a readable way object types, relationships and semantic constraints of this real world. (ii) The logical level is a mapping of the conceptual level into a given type of technological model (relational, object oriented or any other one) without being constrained by a specific platform. Mapping to a relational model does not impose the use of a specific DBMS but a class of DBMS:s. It's main goal is to provide a sound technical description which will serve as a reference to programmers. (iii) The physical level is an implementation of the logical level under certain technical constraints imposed by a specific platform and a specific DBMS, and complying with other user requirements such as performance, security, integrity and so on.

- The second feature is *data-process separability*, which means that the design process of a database is independent of the programming process of applications, each being driven by its own models and design rules (e.g. relational model and normalization on one hand, and Petri nets with formal validation on the other hand). Separability does not mean independency; a given database is built to satisfy some functional requirements. However, it is not necessary to list all user transactions, a global definition of users' needs may be sufficient to proceed. Nevertheless, this separability has been later criticized by the object-oriented community, which insists on dealing with both aspects at the same time (object types are uniformly defined with attributes and operations). But in the database context, this separability principle has contributed to isolate persistent objects and to apply to them specific design models and techniques.

- The third feature is *modularity of the design*. As we have seen earlier, in the context of a distributed information system, there might be different realms which describe data sources, the global schema and the client schemas. The design should consider, at different abstraction levels, each of these realms and explicitly specify how they map to or integrate with each other. Schema integration is one of the most popular techniques in database design; it allows to decompose a complex domain into subdomains, to structure each subdomain and to integrate all of them into one global conceptual schema.

Following the core technology progress, the database design methodology has evolved over the years, gradually mastering complexity and scalability. It was also fed by advances made in Artificial Intelligence and Software Engineering. Depending on the nature of the target information system, the design approach may be simple or complex, short or long, sequential or iterative. Some application domains are complex, others are simple. Some databases are built from scratch, others are built from legacy files or databases.

Schema mapping is the fundamental technique used throughout the database design process. Indeed, from the early design of user views to the final database implementation, the main design tasks consist in transforming a given specification into another. The conceptual-logical mapping is essentially a transformation of a schema defined in a semantic model to a schema defined in another model. Relational normalization is also a mapping process which transforms a given relation into a set of projections satisfying some consistency and redundancy rules. The logical-physical mapping is also a transformation process which maps, for example, a normalized relational schema into an optimized one by denormalization and partitioning techniques.

Schema mapping is based on the *semantic lossless principle*, that is a mapping of a given schema does not lose the semantics captured by this schema. Within the same model, semantic losslessness can be captured by equivalent schemas. For example, in the relational model, two relational schemas are equivalent if the transitive closures of their respective functional dependencies are the same and if the natural join of their respective relations are equal. This is what was used in the normalization process to prove that a normalized schema is equivalent to the former one. In a semantic data model, two schemas are equivalent if there exists a set of reversible rules which allow to derive any of the schema from the other. One can also use a transformation to another model (pivot model): two semantic schemas are equivalent if their mappings to a pivot model produce equivalent schemas.

Finally, we cannot conclude on mapping issues without mentioning that the mapping process is not deterministic by itself. Many candidate rules can be applied at each step and for the transformation of each concept. It is the responsibility of the database designer to make the relevant design choices with respect to his application and his environment. Some of these design choices can be formalized as expert rules and heuristics, but others remain on the designer's or the administrator's hands. This problem results in a new issue which has also been considered: the quality of a schema at different abstraction levels. If the quality of a relational schema at the logical level is clearly stated by normal forms, there is no agreement on a good conceptual schema and a good physical schema, especially when the models used lack of formal foundations. Generally, depending on the model used, and sometimes on the context of use, some quality factors have been proposed to evaluate the human design or the mapping process.

4. The Link with MDA

Database technology and database design were both founded on a model-driven approach. We summarize hereafter analogies and differences with MDA and highlight some database features that can benefit to MDA.

- At the core technology level, the first difference is that the database logical and physical levels represent only data and queries, while MDA is supposed to represent all processes of a given system. PIM:s and PSM:s correspond respectively to logical and physical schemas. As for Database technology, there might be several PSM:s for a given PIM, depending on the platforms or on evolving design choices. Database mappings from the logical level to the physical level are done for data structures and queries respectively at design time (denormalization, partitioning, clustering) and compile time (query rewriting process and execution plan generation). The concept of view is specific to the Database domain, it may correspond in MDA to a restricted PIM describing a subsystem. This leads to the coexistence of a hierarchy of PIM:s, each being mapped to another PIM or to the global integrated PIM. As is the case in database systems, MDA is envisaged within a distributed environment. In this context, there might be several frameworks linking PIM:s to PSM:s, depending on whether one is designing a client-server architecture, a three-tiers architecture or a cooperative system. Software bus based architecture is explicitly assumed to be one of the underlying technologies for implementing MDA. We can easily imagine how the database mediation architecture could be used to implement an MDA approach, hence opening routes to clearly specify the semantics of the different levels and mappings.

- At the design level, the Database design process is probably the one which is the most close to the MDA approach. As it is more mature than MDA, the latter one can profitably benefit from its results and experience. Notions of PIM:s and PSM:s are nothing else than database abstraction levels. However database levels are more precise in their goals and scopes. The conceptual level is an original database notion which has no equivalence in MDA. Mappings within PIM:s or PSM:s and between PIM and PSM are nothing else than schema mappings at the same abstraction level or between abstraction levels. Database models and mappings are almost well established since they are defined on a narrow domain of data modeling. PIM and PSM models are still not fixed, unless UML is considered as a uniform language to describe all abstraction levels. Even in the context of UML, if we suppose that the mappings of class diagrams can benefit from database design, the mappings between PIM:s and PSM:s are still not yet defined, especially for the 8 other diagrams of UML. Again, if the concept of database view can be related to the concept of software component, the integration process is much better fixed in database design than in software component composition, where the problem is certainly harder. Finally, MDA should not be restricted only to software design, it should also include the database design in order to achieve a high level methodology which will be able to effectively help in the design process of a complete information system.

5. Concluding remarks

The purpose of this short paper was to show that Database technology has explored many aspects of model-based design and development. As such, the large amount of expertise and more than 30 years experience accumulated in this area can inspire the progress of MDA. The database community was fed by many ideas from Artificial Intelligence and Software Engineering research and practice, but has fertilized them for a specific purpose of data representation and management. The MDA community should also capitalize from the Database field and address hard problems such as software architecture design, composition of (web) services, model mappings and validation techniques.

MDA for Distributed Real-time Embedded Systems: experience and deployment at CEA

François Terrier, Sébastien Gérard

CEA / DRT-LIST / DTSI / SLA / L-LSP - F-91191 Gif sur Yvette Cedex France
{Sebastien.Gerard, Francois.Terrier}@cea.fr - http://www-drt.cea.fr

1 Why MDA for DRES?

The LSP (Software for Process Safety) Laboratory at CEA (French Atomic Energy Commission) conducts studies on new methods and techniques to ease the development of distributed, real-time, embedded systems (in short DRES). Through regular collaborations with major industrials (e.g.: PSA, Thales, EdF, EADS, CS-SI…) centering its activity on their practical needs, the LLSP has gained a solid expertise on four complementary and essential facets: (i) safety oriented design method and implementation technologies for the development of critical real time systems; (ii) formal specification analysis and automatic generation of test sequences; (iii) methods and techniques for object oriented development of concurrent systems; (iv) implementation platforms for parallel and distributed real time systems.

Among needs expressed by LLSP research partners strong emphasis is made on assisting the management of: system complexity increase, implementation variability, functionality changes and time to market decrease. In addition, strong attention is paid on capturing and generalizing company development know-how in order to introduce and increase reusability not only at the last implementation level but also along the entire life cycle of the system development.

Because, DRES requires precise and complete specification to ensure efficient and reliable implementation they provide naturally very favorable cases to study automation of development steps or of model analysis, e.g.:

1. They intrinsically require various points of view (e.g.: Functional, Real-Time, Security, fault-tolerance…) but separation among product line generic requirements and particular application requirements is difficult (by example due to a priori implementation choices or constraints, or because requirements are often expressed using particular real time values…).
2. Target implementation choices are very variable: they can use various execution models for a same specification, (e.g.: multitask models with RT-OS, Synchronous models, loop programming models, etc.); they can be mapped on various deployments and platforms (e.g.: Single/multi processors, shared/distributed memory…); they use largely proprietary and ad hoc solutions and benefit only from few usable standards.
3. Performance is often very sensitive and, then, classical software engineering techniques of encapsulation could become impracticable (e.g.: multi-level interfaces are often inefficient) and lead to implementation with strong interleaving of real time and functional code. This last point is quite critical when trying to define component models for DRES without knowing how real time code and functional code are intricate inside the component.
4. They are often critical on testing or validation and requires for that both dedicated models and intensive use of system analysis techniques.
5. Generally they require high skill developers (design, implementation, validation) having lot of expertise.

All of these needs on the system development and validation techniques push to introduce intensively MDA techniques along the entire process. They use different and complementary artifacts of MDA that have been experimented, implemented and evaluated in the ACCORD/UML methodology tool kit as illustrated below. For more details on the ACCORD/UML methodology, interested readers may have a look on [1-11].

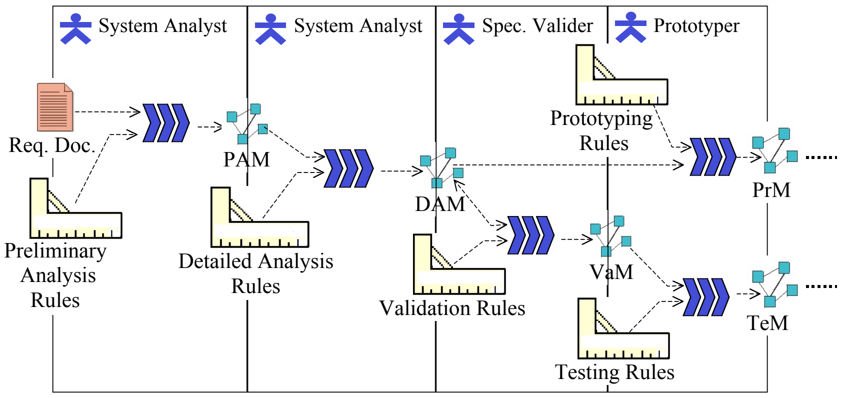
2 What has been done at CEA?

The ACCORD/UML methodology targets initially engineers who are not "software" experts. It aims at supplying a method and its underlying tools with a sufficient level of abstraction to specify and prototype DRES without requiring being a real-time expert. By this way, they may build "software-intensive" systems without being software experts, and in particular "DRES software experts", in order to concentrate their activity on their main business.

2.1 Development process models

To support the methodology users require ways to organize the development process in steps (point related to the SPEM Profile) and to formally: define the different types of models (or "views" of the global and complete system model); define composition ("weaving") rules, filtering rules, mapping rules, etc. Purpose of that is to be able to have all the information, and only them, required describing and understanding the system for a given point of view. This will ease development of model ensuring a high level of separation of concerns.

As depicted in the figure, ACCORD/UML relies on a basic three stages software development cycle: analysis, prototyping and testing. Progression from one to another phases is achieved by a continuous and iterative process of refining UML models based on different sets of mapping rules. For that purpose, specific rules mapping packages has been specified within the ACCORD/UML profile [12].



During analysis, the system is first considered as a black box focusing the modeling activity on its interaction with its environment and then it is refined describing inside the black box the structural, interactional and behavioral aspects of the application. However, in this step, models must make no reference to any implementation solution. This process illustrates the importance of managing several views of the system (here along its life cycle) and to be able to attach to each kind of view a set of modeling rules ensuring its is built consistently and it remains in the objective of the process activity in which it is elaborated.

2.2 *High level modeling concepts*

To ease the expression of the requirement and the first step of DRES model elaboration, it is important to provide ways to introduce abstraction dedicated to the domain such as “Real Time Object” (point related to standard extension mechanisms: stereotypes, tagged values, OCL constraints…), “Real Time Feature” (point related to the SPT Profile [13] and to the under work QoS & FT Profile[14]). The goal of these specializations is both to allow the user manipulating simple concepts and to hide implementation details that will be described and decided in later steps during the development process. These models refer to PIM models of the system. One of the major features of MDA approach is the possibility through model transformations to define heuristics implementing these high level abstractions without paying too much on final performance. Existence of this possibility favors the acceptation of these high level concepts by DRES developers. Among the UML extensions introduced by ACCORD/UML, we can mention the two following that illustrate various ways to use these mechanisms:

- “Real Time Object” stereotype that encapsulates tasks and concurrency management and suppresses any reference to RT implementation in code.

- Real time quality of service specializing the SPT Profile to manipulate concretely specifications assigned to RTO messages in terms of deadlines, periods, ready times, etc.. Explicit use of this element allows automatic extraction or update of real time requirements and provides models ready for scheduling, performance analysis.

All UML extension definitions, introduced to make accessible to non-experts the real-time systems development, are also contained in the ACCORD/UML profile.

2.3 *Execution models*

PIM of DRES can be map to different Computation Description Model (CDM) modeling various execution models depending on implementation constraints, application requirements and internal company strategies. The separation of concerns between a system model focused on the application functions and requirements and the system model integrating a particular execution model is very important for the potentiality of reuse of the application. These models are independent to an implementation on a precise target (there are themselves PIMs), but when attached to the system model to describe choice on its execution principles they made implicit assumption on the underlying implementation platform and should be considered as a high level PSMs.

2.4 *Implementation platform models*

Models of implementation platforms complete the execution models by defining the targets on which they will execute. Then, the system model, including its execution model, can be attached (mapped, composed, weaved…) to these implementation models in order to provide a PSM of the system. These models provide users with dedicated view of the platforms interface (e.g. UML profiles for CORBA, CCM, etc…). Their goal is to present to the user a minimal view of the platform specifications required by the current stage of the development process.

2.5 Model analysis and transformation techniques

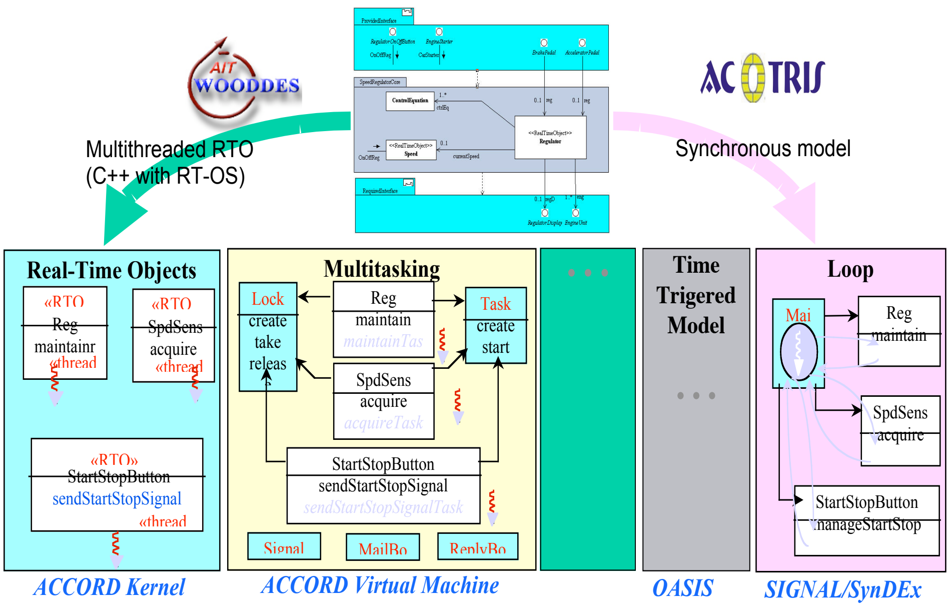
Within a MDA context, to refine models, i.e. to go from one abstract model to more concrete, a key point is focused on mapping issues. The different possibilities of mappings defined in [15] are: PIM □ PIM, PIM □ PSM, PSM □ PSM and PSM □ PIM. In the UML world, it is possible to distinguish between two kinds of mapping:

- Endomorph transformation: the source and target models, either PIMs or PSMs, are relative to the same meta-model, i.e. the UML one. This is the case, in particular, during the development process for the refinement of the model from one step to the other, but this is also the case when extracting, for test generation purpose, a behavior model issued from a couple of execution model and business model.
- Exomorph transformation: in this second case, the source and target models are heterogeneous. This means that the source or the target model are relative to different meta-model than the UML one. An example of such mapping is the usual code generation that may be observed as being a transformation from a UML-based PIM or PSM into a C++ based PSM for example, but this is also the case, for test generation purpose, when exporting the model to a tool using an other formalism and importing results from this tool into the UML modeling environment.

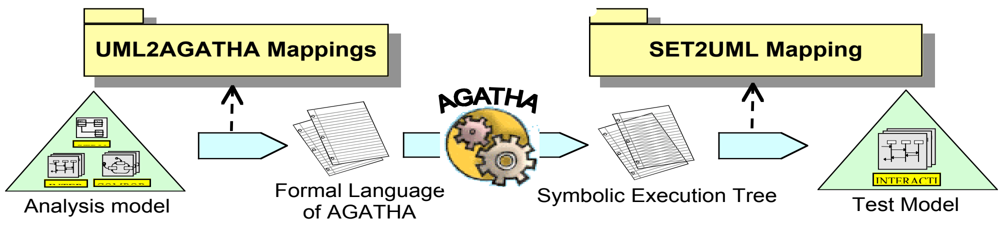
As mentioned previously, the choice of the execution model used in ACCORD/UML is performed through implementing particular design patterns and code generation procedures. Although a high level mechanism would ease to refine and reuse these mechanisms or part of them, this approach allows generating from the same high-level model (PIM) two different implementations:

□ First runs on real-time operating systems with real time objects has been experimented on an automotive case study during AIT-WOODDES project (http://wooddes.intranet.gr/);

□ Second is based on synchronous models and runs on multiprocessors machines without requiring any operating systems is experimented on two case studies of the ACOTRIS project (www.acotris.c-s.fr), parallel image processing and parts of a distributed I&C of a liner.



In practice, these transformations formalize the implementation expertise and allow reusing them on several applications. Also in the context of exomorph transformations, the ACCORD/UML approach provides the capability to automatically produce test sequences form the system model. This relies on a model automatic analyzer called AGATHA [16]. It is a formal-based tool taking as entry a low level formalisms based on an Extended Input Output Labeled Transition Systems. Connecting this tool to a UML modeler requires model transformations procedures that reformulate the application model with its chosen execution model in terms of a set of concurrent automatons communicating via rendezvous. Reverse transformation is also required to traduce the formal execution sequence into UML sequence diagrams representing the test cases.



3 How MDA would help for DRES?

Five points have been listed regarding needs for DRES development. The ACCORD/UML experience shows that they could be largely covered via techniques related to MDA and development of dedicated MDA artifacts:

1. *They intrinsically require various points of view*: process definition and tooling associated to UML extensions providing high level concepts can formalize content of the models, their interdependency and allow to make visible on them requirements depending on implementation concerns with possibility to extract them easily or modify them without changing the rest of the models.
2. *Target implementation choices are very vari*able: definition of Computation Description Models and of Platform Description Models allows to separate these elements to the rest of the application model. Dedicated transformations can, then, take in charge a large part of the final implementation effort.
3. *Performance is often very sensitive*: Code generation heuristics have shown that it is possible to both manipulate high level concepts at the model level and to produce efficient implementation suppressing the usual overheads obtained when implementing and running these concepts at the execution level.
4. *They are often critical on testing or validation*: definition of UML extensions suppressing ambiguities and ensuring semantic completion can allows the development of mapping procedures to the application models on formal models used by formal analysis tools in order to derivate test sequence or obtain feasibility of the scheduling.
5. *Generally requires high skill developers*: generic implementation patterns and architectures are largely used for DRES. They can be introduced in the development process as reusable pieces together with the dedicated transformation procedures. They allow easy reuse of developers' know-how.

The question of developing a set of MDA components for development, validation, exploitation and maintenance of DRES is on the center of a new large joined research collaboration among CEA, INRIA and Thales: the CARROLL program (www.carroll-research.org).

4 Missing pieces and conclusions

Models of DRES consist then of three main pieces: models describing business views (PIM) of the application; models clarifying the computation model (CDM) used implicitly to model business views; models describing the targeted platform (PSM) for the system implementation. DRES engineers will then require standardized CDMs and PSMs in order to be able implement their various applications available under the form of PIMs. Moreover CDMs will be more and more important in the context of model transformation. Until now, model transformation validation is focused on syntax point of view but when it is requires to check semantic invariance of model mapping, there are not yet clear answer to this issue. Availability of such CDMs should ease solving this point, because simply execution model will then be explicitly described. However, MDA concepts such PIM, PSM, PDM still requires clarification and clear definition [17].

The second point relative to MDA success will be the availability of a mapping or transformation language. The current work starting by OMG through the RFP Query, View and Transformation (Q/V/T) should answer this point. But following the example of the Action Semantics of the UML, there is a risk that people working on it may not find an agreement on a single mapping language. For example, it will probably impossible to impose the language to be imperative or declarative. So it is possible that the answer consists of different mapping languages. In this context, the inter-operability of each proposal will be a mandatory requirement in order MDA to be a fully efficient.

Finally, going from code-oriented to model-oriented development will not avoid us to solve usual problems such as requirement traceability, configuration and version management, … So in order MDA to be a success in industry, solutions to these issues will have to be available in the tools stating to be MDA-compliant!

5 References

1. A. Lanusse, S. Gérard, and F. Terrier. "Real-Time Modeling with UML : The ACCORD Approach", in "UML'98 : Beyond the Notation". 1998. Mulhouse, France: J. Bezivin et P.A. Muller.

2. S. Gérard, N.S. Voros, C. Koulamas, and F. Terrier. "Efficient System Modeling of Complex Real-time Industrial Networks Using The ACCORD/UML Methodology", in Architecture and Design of Distributed Embedded Systems (DIPES 2000). 2000. Paderborn University, Germany: Kluwer Academic Publishers.
3. S. Gérard, Executable modelling for automotive embedded systems development, PhD Thesis, in GLSP. 2000, Evry University, Paris.
4. F. Terrier and S. Gerard. "For a full integration of real-time concern into OO models, or "How to popularize real time programming ?"", section in "Real Time UML: Heaven or Hell" Panel in UML'2000 proceedings. York, UK: Springer.
5. F. Terrier and S. Gérard. "Real time system modeling with UML: current status and some prospects", in 2nd Workshop on SDL and MSC 2000. 2000. Grenoble, France.
6. S. Gérard, E. Pelachi, P. Petterson, and B. Josko, "Methodology for developing real time embedded systems", CEE: Pub, D3/CEA/WP1.3/V1.0. 2002. p. 158.
7. S. Gérard, F. Terrier, and Y. Tanguy. "Using the Model Paradigm for Real-Time Systems Development: ACCORD/UML", in OOIS'02-MDSD. 2002. Montpellier: Springer.
8. S. Gérard and F. Terrier, "UML for Real-Time", chapter in "UML for Real: Design of Embedded Real-Time Systems", L. Lavagno, G. Martin, and B. Selic, Editors. 2003, Kluwer Academic Publishers: Boston. p. 369.
9. N. Guelfi, A. Schoos, S. Gérard, and F. Terrier, "EUDEMES: Component-Based Development Methods for Small-Size Embedded Systems", ERCIM NEWS, 2003. vol. 52 (Embedded Systems): p. 64.
10. C. Mraidha, S. Gérard, F. Terrier, and J. Benzakki. "A Two-Aspect Approach for a Clearer Behavior Model", in The 6th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'2003). 2003. Hakodate, Hokkaido, Japan: IEEE.
11. P. Tessier, S. Gérard, C. Mraidha, F. Terrier, and J.-M. Geib. "A Component-Based Methodology for Embedded System Prototyping", in 14th IEEE International Workshop on Rapid System Prototyping (RSP'03). 2003. San Diego, USA: IEEE.
12. S. Gérard, "The ACCORD/UML profile", CEA-LIST: Internal report. 2002.
13. OMG, "UML Profile for Schedulability, Performance and Time (ptc/02-03-02)", OMG, ptc/02-03-02. 2003. p. 154.
14. OMG, "UML Profile for Modeling QoS and FT Characteristics and Mechanisms RFP", OMG, ad/02-01-07. 2002.
15. J. Miller and J. Mukerji, "Model Driven Architecture (MDA)", OMG: Specification. 2001. p. 31.
16. D. Lugato, N. Rapin, and J.-P. Gallois. "Verification and tests generation for SDL industrial specifications with the AGATHA", in Workshop on Real-Time Tools, CONCUR'01. 2001.
17. J. Bézivin and S. Gérard, "A Preliminary Identification of MDA Components", OOPSLA 2002 Workshop: Generative Techniques in the context of Model Driven Architecture. 2002.