Practical Experiences in the Application of MDA

Miguel de Miguel[2], Jean Jourdan[1], Serge Salicki[1]

[1] Thales-TRT
Domaine de Corbeville 91404 Orsay, France
{Jean.Jourdan,Serge.Salicki}@thalesgroup.com
[2]E.T.S.I. Telecomunicación
Ciudad Universitaria, 28040 Madrid Spain
mmiguel@dit.upm.es

Abstract. During the last ten years a lot of concepts have emerged from software engineering. Most of them try to formalize and organize software engineering knowledge at a higher level of abstraction, giving code a secondary role. Model Driven Architecture (MDA), putting the concept of the model on the critical path of software development, is about to change this situation, turning the role of models from contemplative to productive. This paper introduces some problems detected in the process of adoption of MDA methods. The application of MDA for a specific domain, technique, or technology requires the *Description of Specialized Modeling Language*. Two main techniques provide support for the description of UML extensions, *MOF meta-models* and *UML profiles*. The *Process of Mapping Description* requires good support to identify the elements of the source modeling language that are mapped and the model elements of the destination modeling language that correspond to the source elements. This process require specific solutions.

1 Introduction

During the last ten years a lot of concepts have emerged from software engineering. Most of them try to formalize and organize software engineering knowledge at a higher level of abstraction, giving code a secondary role. Among others, the concept of software architecture, architectural patterns, design patterns, idioms appear to formally and explicitly support the different knowledge of development know-how. They are mostly used as guidelines and guidebook to help the development of new applications. MDA [2], putting the concept of model on the critical path of the software development, is about to change this situation, turning the role of models from contemplative to productive. The goal of model driven engineering is to define a full life cycle method based on the use of various models automating a seamless process from analysis to code generation. This discipline puts in the right place all the software artifacts (e.g. business models, architectural models and design patterns) and uses them actively in order to produce and deploy an application.

MDA pays special attention to the *separation of technology dependent concepts from independent concepts*. This solution limits the problems of platform dependencies and portability of the software. The separation is supported at model level to avoid platform dependencies in all phases of the life cycle. The Platform-Independent Model (PIM) to Platform-Specific Model (PSM) mappings provide support to reduce the cost of adaptation of the subsystems to different platforms. Sometimes the allocation of subsystems in a specific platform cannot be decided in early phases, and this mapping reduces the problems of reallocation of subsystems in other platforms. During last year, middleware platforms (i.e. CORBA, DCOM, and Java RMI), component platforms (i.e. CCM, EJB, and .NET) and object-oriented programming languages have appeared and none of them prevails over the others.

MDA proposes solutions to *automate the software development process*. The main objective is the reduction of the time to market based on tool support for the refinement of models and code generation. This approach reduces development errors because it reduces the manual development process and provides support to reuse the best-known solutions. In this development process, the tools can provide support for the integration of different software development phases based on the transformation of models of different phases. The tool support provides a constructive method based on models with the combination of concerns at modeling level.

Models provide support for different types of problems: i) description of concepts, ii) validation of these concepts based on checking and analysis techniques, iii) transformation of models and generation of code, configurations, and documentation. *Separation of concerns* avoids confusion because of the combination of different types of concepts. MDA introduces solutions for the specialization of the models for specific concerns and for the interconnection of concerns based on models. This approach reduces the complexity of models by the specialization of modeling activities. It improves communication between stakeholders using the models to support the interchange of information.

This paper introduces some problems detected in the process of adoption of MDA methods. During last years, Thales has started some actions for the adoption of model driven engineering techniques. Two pilot programs have UML and modeling engineering as main references in the software development process. These pilot programs include the application and evaluation of these techniques in four real projects, and other actions develop supports for the adoption of these techniques and for their practical application in the near future. The real projects use current solutions for the transformation of models and code generation and evaluate the cost of their application. And the innovation actions include the development of UML extensions to support specific domains modeling languages, techniques, and technologies, and the models transformations [25]. This paper presents the results of the evaluation of current solutions for the application of MDA solutions, and the subjects that we have identified as critical in their adoption.

The rest of this paper introduces some related work (Section 2). Section 3 describes basic concepts of MDA. Sections 4 and 5 present some problems and solutions for the adoption of MDA, and Section 6 presents some conclusions.

2 Related Work

In March 2001, the Platform and Domain Technical Committees of OMG endorsed MDA [22][21][6]. [2] defines a model as "a formal specification of the function, structure, and/or behavior of a system." Within the MDA, an interface standard – for example, a standard for the interface to a manufacturing execution system – would include a PIM and at least one PSM. The PIM captures the conceptual design of the standard, untainted by the special features or limitations of a particular software technology.

Some of the motivations of the MDA approach: i) reduce the time of adoption of new platforms and middleware, ii) primacy of conceptual design, and iii) interoperability. The MDA approach makes it possible to save the conceptual design, and the MDA helps to avoid duplication of effort and other needless waste [7][19].

The MOF (Meta-Object Facility)[12] modeling stack, the model transformation process based on meta-model specifications and model serialization is a solution for the support of model-based development process [3]. The transformation of the UML model is specified by a set of rules defined in terms of the corresponding meta-models. Nordmoen in [11] introduces the application of Model Driven Development by applying formal models on all levels of system-analysis, which are reused at different levels of abstraction. This makes it easier to remodel a system to fit changes in business needs. They separate platform independent concepts from specific concepts and improve the potential for faster development by supporting code generation and communication middleware independence.

Some approach proposes a language for object-oriented modeling [4]. The language provides support to describe well-structured modeling languages and integrates the semantic of OCL. *USE* is a tool for the validation of UML models and OCL constraints [20]. But in this approach, the validation of constraints is supported by the execution of models.

Some tool providers [23][9][10] propose different solutions for the adoption of MDA. Each solution proposes specific solutions for the implementation of model transformations (i.e. JPython, Action Semantic [17] and J languages), and different solutions for the specification of modeling elements of transformation processes (i.e. proprietary modeling languages, meta-models and UML profiles).

3 MDA Overview

MDA provides a solution for the software development process based on models. The models can be looked at from different points of view (e.g. problem domains, architectural solutions), studied for different purposes (e.g. analysis of problems, evaluation of architectural solutions), and their evolution and transformation can address different objectives (e.g. integration of technical concepts, transformations between different programming language domains). UML is the modeling language reference of MDA, but it is a general purpose modeling language that must be extended and adapted for specific purposes.

Figure 1 introduces the most important concepts of MDA technology. PIM concepts are based on three different types of UML modeling points of view: i) *Domain Modeling*, ii) *Architectural Modeling,* and iii) *Technical modeling* based on specific types of services. These types of models pay attention to some specific concepts and provide extensions and specializations of UML to support these concepts. PSM models are based on *technological models* that provide support to model in UML platform specific concepts. The specific PSM modeling notations are based on semantics and concepts of execution infrastructures. Mapping facilities provide the support for the transformation of models, and they support the evolution of models.

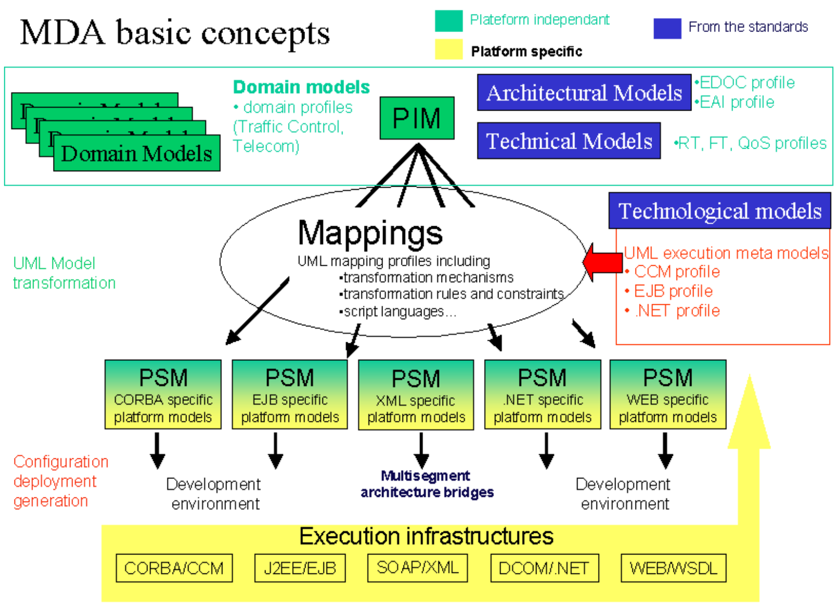
MDA basic concepts
Plateform independant
Platform specific
From the standards
Domain Models
Domain models
• domain profiles (Traffic Control, Telecom)
PIM
Architectural Models
•EDOC profile
•EAI profile
Technical Models
•RT, FT, QoS profiles
Technological models
Mappings
UML mapping profiles including
•transformation mechanisms
•transformation rules and constraints
•script languages...
UML Model transformation
UML execution meta models
• CCM profile
• EJB profile
• .NET profile
PSM CORBA specific platform models
PSM EJB specific platform models
PSM XML specific platform models
PSM .NET specific platform models
PSM WEB specific platform models
Configuration deployment generation
Development environment
Multisegment architecture bridges
Development environment
Execution infrastructures
CORBA/CCM
J2EE/EJB
SOAP/XML
DCOM/.NET
WEB/WSDL

Fig. 1. MDA Basic Concepts

3.1 MDA models and viewpoints

MDA identifies three main areas of UML meta-model extension and specialization:

1. Domain extensions. Examples of domains for the application of UML modeling are: telecom, e-commerce and healthcare. These domains use specific modeling structures and methods and require specialized modeling elements to improve description domain dependent concepts.

2. Technical and architectural extensions. Some common techniques are used in different domains. Some examples are: real-time, fault-tolerance, transactions and security. These techniques require UML extensions to represent technical concepts not supported in UML (e.g., specific temporal properties [15] and complex identification methods). The technical concepts have associated specific types of architectural concepts (i.e., distributed executions [13]).

3. Technological extensions. These extensions provide support to express in UML specific programming language and middleware concepts. Some examples are the profiles to express in UML, EJB [24] and CORBA [14] concepts.

4 Implications in the Adoption of MDA

The application of MDA requires the description of domain, technical and technological modeling extensions, and the specification of mappings between modeling extensions.

4.1 Description of Specialized Modeling Languages

The definition of new UML extensions can be a complex process for two main reasons: we must explicitly state the extension concepts and their specific properties, and we must identify what the relations are between these concepts and UML meta-model elements. In general, the concepts of *domain* and *technical* extensions are well known in these environments but do not have a precise specification. *Technological* extensions (e.g., middleware and programming languages) have a formal specification in terms of grammar, but the integration of their detailed concepts in UML notations is a complex process.

Two main techniques provide support for the description of UML extensions:

• *MOF meta-models*. The definition of a new extension based on MOF requires: i) the definition of the extension meta-model based on MOF meta-meta-model notations (static diagrams and *well-formed* OCL rules), ii) the definition of the subset of UML meta-model (which may be the entire UML meta-model), where we will apply the extensions, iii) then we must integrate the extension meta-model with the UML meta-model. The integration defines the relations (e.g., associations and inheritances) between extensions meta-classes and UML meta-classes, and iv) definition of new *well-formed* rules in OCL related with the new relations and the combination of both meta-models. The result is a new meta-model that includes and integrates both meta-models.

Some infrastructures for the adoption of this approach include: i) capabilities for the **definition of meta-model** (tool support for the specification of MOF models), ii) **specification and evaluation of well-formed rules** (the validation of meta-model constraints), and iii) **extension of UML meta-model** (the new meta-models include associations with UML meta-model elements).

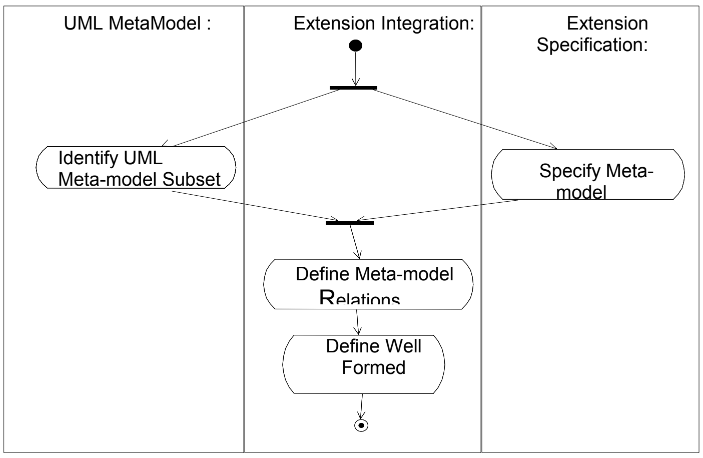

Fig. 2. Extension Based on Meta-Models

- *UML profiles*. An extension based on profiles requires: i) the definition of the extension meta-model, when the extension concepts do not have a formal definition (in general, domain and technical extensions). This meta-model, like in the previous case, will provide the reference of the extension concepts. When the extension is based on a formal defined terminology (e.g., the subset of a programming language specification or middleware standard), we must specify the subset of the extension space. ii) The definition of the subset of UML meta-model (which may be the entire UML meta-model), where we will apply the extensions, iii) the specification of UML extension elements (stereotypes, tagged values and types of constraints) that will support the extension meta-model, iv) for each stereotype and tagged values or constraints not associated with stereotypes, we must define the UML meta-model elements that can have associations with these extension elements. And v) the definition of the *well-formed* rules for the UML elements that can have the extension elements associated. The result is an UML profile that provides support to extend UML models with standard extension techniques. Some tools provide **profile support** based on profile builders, and facilities to **reuse the profiles** in the construction of new profiles.

Both extension techniques have advantages and disadvantages. MOF meta-metamodel language provides considerable support for the *description of relations* (i.e., associations and generalizations) and conceptual elements (i.e., classifiers, features and packets). UML extension mechanisms are limited to stereotypes, tagged values and constraints and their relations are limited to stereotype inheritance and aggregation of tagged values in stereotypes. Because MOF meta-metamodel language includes different types of relations, these relations can be used in the *combination of two meta-models* (i.e., inheritances and associations). The integration of UML extension mechanisms is based on the association between UML meta-classes and the new stereotypes and tagged values, this does not allow describing new associations and inheritances in UML meta-classes.

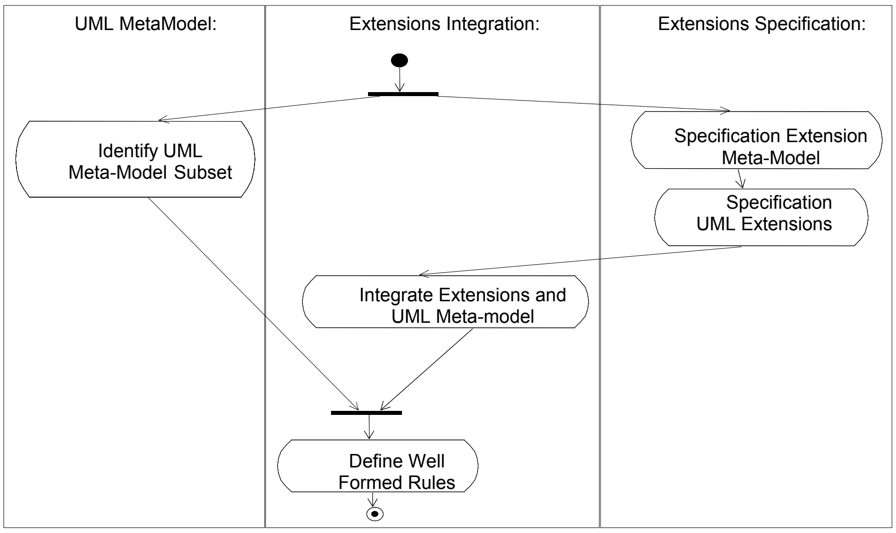
UML MetaModel:
Extensions Integration:
Extensions Specification:
Identify UML Meta-Model Subset
Specification Extension Meta-Model
Specification UML Extensions
Integrate Extensions and UML Meta-model
Define Well Formed Rules

Fig. 3. Extension Based on Profiles

Currently, in general, UML tools provide the *support of UML extension mechanisms* (stereotypes, tagged values, and constraints). New extension based on them can be integrated rapidly in these tools. OMG and other organizations have developed some *UML profile standards* and some standards are in progress. The standardization of general extensions is required to make the models tool independent and to interchange models among different tools. Sometimes we require applying *different extensions to the same business model*. Two different profiles can be applied to the same model. But the application of two MOF meta-models require their combination to create a third meta-model, and then the integration of this with UML meta-model.

4.2 Description of Model Transformations and Mappings

The methods for the description of modeling languages and extensions are a basic support for the description of model mappings and their semantic. The mappings provide the support for the MDA model-based development process. **Supports for the mapping description** identify the elements of the source modeling language that are mapped and the equivalent of destination, and the guards that the model must fulfill to apply this type of transformation. Currently the mapping process is put into practice in different ways:

1. *Script languages*. Some UML tools include imperative script languages with meta-model navigation facilities similar to OCL navigation expressions. These languages are the support to implement the mapping scripts. These languages are

flexible but often are tool dependent, and a **mapping implementation is not portable**.

2. Some tools provide support to process and generate *XML and XMI files*. These files include the meta-data of UML models and mapping is supported by transformation of *XMI* files. The mapping is independent of the UML tool.

3. Some tools include *MOF transformation* facilities based on rules. These rules provide facilities to identify the elements in the source model, where we apply the rule and the destination elements that we generate with the rule.

Some infrastructures include facilities for the automatic **generation of new models**, these facilities include the creation new modeling elements and diagrams and their integration in the repository.

MDA requires model transformation, which is a basic tool for the evolution, specific analysis, and connection with implementation platforms of models. Two main concepts are involved in model transformation: the source and destination modeling languages and the mapping between languages. UML transformations are used for three general purposes:

1. UML Model transformations and refinements. MDA proposes the refinement and transformation of models as a basic technique to extend or specialize a model. PIM models are transformed into PSM to introduce platform specific concepts not included yet in the platform independent model. Some specific concepts are introduced in the generated model automatically and others are updated manually. PSM to PSM and PIM to PIM refinements provide support to improve a model in the same modeling language space.

2. UML model evaluations. Some UML technical profile standards and RFP [15][16] provide UML extensions and support for the transformation of UML extended models into other types of modeling techniques (e.g. analysis methods like *Rate Monotonic Analysis* and *Failure Mode Effects Analyses*) to apply specific analysis methods. Some UML tools make the transformation of UML models into simulation models to do the evaluation of the original models.

3. *Generation of implementations.* Generators that provide as a result platform specific implementations can support the implementation of a PSM model. These generators translate the UML model into programming language and middleware constructors (e.g., Java and CORBA interfaces and EJB component descriptors).

The transformation and refinement process include the problem of **traceability**. It is generally recognized that UML's facilities for relating models at different levels of abstraction are rudimentary and need expansion. The UML 2.0 includes this as a basic problem. Another problem is the capacity to **reuse the mappings** and the combination of different mappings to create more complex mapping. This is especially interesting in MDA because it describes a sequence of mappings applied at different levels of abstraction.

5 Solutions for the Adoption of MDA

In Section 2 we have introduced some approaches to support MDA. In this section, we are going to give five different solutions based on these approaches and describe how they support different facilities introduced throughout the paper. The five approaches are:

- **Integration of Generators** [9][8]. proposes some pragmatic solutions for the application of MDA. *ArcStyler* includes support for the development of component-based architectures implemented on EJB platforms. Business and platform independent concepts are expressed on business models. Business modeling comprises the first stage in the full cycle development of component-based software systems. These models are transformed automatically into component models, which are EJB specific models. Business models are expressed in a proprietary modeling language, and component models are based on a UML profile. *ArcStyler* provides support for the automatic generation of EJB components, from component models. *ArcStyler* supports customization of the transformation process. Customization is based on the cartridge configuration files and templates. JPython is used as the transformation language, some scripts are generated automatically, and others are specific to the generation process.
- **Profiling** [23][5]. *Objecteering* provides support for the description of UML profiles (*Profile Builder*). The profile description includes the specification of *Stereotypes* and *Tagged Values* and the UML meta-classes associated with these extensions. The profiles are supported and handled with *modules*. A module can include commands applied to model elements. These commands implement the transformation of models. They are scripts in a proprietary language (J language). J provides support for the creation of new diagrams and model elements. The commands support transformations from model to model, or from model to code or documents. *Objecteering* provides traceability support to avoid inconsistency between the model source and the model or code destination.
- **Execution of Models** [10]. iUML comprises a modeler and simulator. The iUML simulator provides an execution environment in which models can be executed and supports Action Specification Language (ASL). iUML supports pre-defined mappings to platform specific implementations, and the definition of user configurable mappings from PIM to specific implementations. The mappings are specified using executable UML models that represent the source and destination meta-models. In this approach, meta-models and ASL support the mappings, and the execution of UML models at meta-model level implements the transformations.
- **Languages for Modeling** [1][4]. The 2U group proposes a meta-modeling framework. The facility comprises a language for defining modeling notations, a tool that checks and executes those definitions, and a method consisting of a model based approach to language definition. Mappings between language components are defined in terms of OCL constraints on associations between model elements. Currently, there is no tool support available for this solution.
- **OCL Evaluators** [26][20]. USE is a tool for the validation of UML models and OCL constraints. This tool does not address the problems of model mappings and UML extensions but proposes solutions for the validations of OCL constraints. This approach is a solution for the validation of OCL constraints in mappings and meta-models.

Tab. 1. Comparison of Solutions for the Adoption of MDA

			Integration Generators	Profiling	Models Execution	Languages for Modeling	OCL Evaluators
Description of Modeling Languages	Profiles	Profile Support	(A)/(D)	AD PM PS	--	(A)/(D)	--
		Reuse of Profiles	--	AD PS	--	--	--
		Specification Well-formed Rules	--	--	--	AD	AD SS
	Meta-Models	Definition of Meta-Models	(A)/(D)	--	AD	AD	--
		Extension UML Meta-Model	--	AD PS	AD	AD	--
Mappings		Mappings Support	MSL	MSL PS	MMM SS	AD	(A)/(D)
		Models Generation	AD PS	AD PS	(A)/(D)	--	(A)/(D)
		Portability of Mappings	--	--	AD SS	(A)/(D)	--
		Traceability	(A)/(D)	A/D	--	--	--
		Reuse of Mappings	AD RT RA	AD PS	(A)/(D)	(A)/(D) RT	--

In Table 1, we present a simple qualitative comparison of the solutions. We use the support facilities we have identified in Section 4, to do the comparison. The legend for included is as follows:

-- "not addressed"	**PM** "Support of Profiles Multiples"
AD "addressed in detail"	**MMM** "Mappings guided by meta-models"
(A)/(D) "mentioned only in the tool board"	**MSL** "Mappings guided by script languages"
NF "To be addressed in next future"	**RT** "Reuse of mappings based on templates"
PS "Proprietary solution/language"	**RA** "Reuse based on annotations"
SS "Standard solution/language"	

6 Conclusions

MDA proposes solutions to separate technology dependent concepts and independent concepts. The separation reduces the complexity of software portability and specializes the modeling process. This approach make easier the application of tool supported solutions.

The adoption of MDA require to take some decisions about the technologies to be used, and the tools and methods that can support MDA. Currently there are different solutions to support the definition of modeling language extensions and there is not a standard available for the specification of mappings.

These problems can create problems equal or more complex than the problems that MDA try to solve. MDA proposes solutions to limit the dependencies of software from the platforms, but the alternative solutions for the specification of extensions and mappings can create dependencies with the specific mapping implementations. If there is not a precise description of mappings, two different implementation of mappings can generate very different code or models and this can create the dependencies between the software and the mapping solution used. OMG has started to address this problem [18].

References

1. 2U Consortium. http://www.2uworks.org
2. Architecture Board ORMSC "Model Driven Architecture", OMG document number ormsc/2001-07-01. (July 2001) ftp://ftp.omg.org/pub/docs/ormsc/01-07-01.pdf
3. J. Bézivin. "From Object Composition to Model Transformation with the MDA". In Proceedings of TOOLS´2001. IEEE. (August 2001).
4. T. Clark, A. Evans and S. Kent. "Engineering Modelling Languages: A Precise Meta-Modelling Approach". http://www.2uworks.org/documents.html
5. P. Desfray "MDA-When a major software industry trend meets our toolset, implemented since 1994". (October 2001) http://www.softeam.org/fr/pdf/mda.pdf
6. D. DSouza. "Model-Driven Architecture and Integration: Opportunities, and Challenges", version 1.1, http://www.catalysis.org/publications/papers/2001-mda-reqs-desmond-6.pdf
7. D. Flater. "Impact of Model-Driven Architecture". In Proceedings of the 35th Hawaii International Conference on System Sciences, (January 2002)
8. R. Hubert. "Convergent Architecture: Building Model-Driven J2EE Systems with UML" J. Wiley (November 2001)
9. IO-software: *ArcStyler*. http://www.io-software.com/products/as_mda_main.html
10. Kennedy Carter: "iUML - The executable UML modelling environment". http://www.kc.com/products/iuml
11. B. Nordmoen "Beyond CORBA Model Driven Development". http://www.omg.org/mda/mda_files/SSSummit_nordmoen_OMG.pdf
12. Object Management Group. "MOF 1.3 Specification", OMG document number formal00-04-03, (March 2000) ftp://ftp.omg.org/pub/docs/formal/00-04-03.pdf
13. Object Management Group. "UML Profile for EDOC". OMG document number ptc01-12-04 (December 2002) ftp://ftp.omg.org/pub/docs/ptc/01-12-04.pdf
14. Object Management Group. "UML Profile for CORBA 1.1 Specification", OMG document number ad/99-03-11 (March 1999) ftp://ftp.omg.org/pub/docs/ad/99-03-11.pdf
15. Object Management Group. "UML Profile for Schedulability, Performance, and Time", OMG document number ptc02-03-02 (March 2002) ftp://ftp.omg.org/pub/docs/ptc/02-03-02.pdf

16. Object Management Group. “UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms”, (Request for Proposal). OMG document number ad02-01-07 (January 2002) ftp://ftp.omg.org/pub/docs/ad/02-01-07.pdf

17. Object Management Group. “Actions Semantic for the UML”, OMG document number ptc02-01-09 (January 2002) ftp://ftp.omg.org/pub/docs/ptc/02-01-09.pdf

18. Object Management Group. “MOF 2.0 Query / views / transformation”, (Request for Proposal). OMG document number ad02-04-10 (April 2002) http://cgi.omg.org/docs/ad/02-04-10.pdf

19. D. Poole. “Model-Driven Architecture: Vision, Standards And Emerging Technologies”. Workshop on Metamodeling and Adaptive Object Models. ECOOP 2001.

20. M. Richters and M. Gogolla. “Validating UML Models and OCL Constraints” In Proceedings of UML 2000. Springer. (October 2000)

21. J. Siegel and the OMG Staff Strategy Group. “Developing in OMG’s Model-Driven Architecture”. Object Management Group White Paper (November, 2001) ftp://ftp.omg.org/pub/docs/omg/01-12-01.pdf

22. R. Soley and OMG Staff Strategy Group. “Model Driven Architecture”, Object Management Group White paper, (November 2000) ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf

23. Softeam. “Objecteering/UML CASE tool”. http://www.objecteering.com

24. Sun Java Community Process JSR-26. “UML/EJB Mapping Specification”. http://jcp.org/jsr/detail/26.jsp

25. Thales-TRT. “Model Based Engineering of Software Products”. In proceedings of OCM 2002 (March 2002)

26. “USE: A UML-based Specification Environment”. http://dustbin.informatik.uni-bremen.de/projects/USE/