

Model-based System Development
www.ifi.uio.no/inf5120

Part I

MDE - Model Driven Engineering

Lecture Notes for Course

"Model Based System Development"

INF5120 – Spring 2008

Classification	Notes for course participants
Project Responsible :	Arne-Jørgen Berre, SINTEF and University of Oslo
Authors :	Arne-Jørgen Berre, Brian Elvesæter
Contributors:	Projects INTEROP, ATHENA, SWING, SHAPE, COIN, Gøran Olsen
Task	INF5120 Course notes
Status :	Version 1.00
Date :	May 2, 2008

Table of Contents

Executive Summary

This document is part I in a series of four part of lecture notes for the course INF5120, Model Based System Development, for the spring of 2008, as follows.

Part I – MDE – Model Driven Engineering
Part II – SOA - Service Oriented Architectures
Part III – MDE for SOA
Part IV – MDI – Model Driven Interoperability

Part I focuses on - *Model Driven Engineering* – with an introduction to principles of metamodelling and rel *MDE* ated standards and technologies, in particular related to MDA and Eclipse EMF. The relationship between UML profiles and Domain Specific Languages (DSL) is introduced, as well as an overview of various model transformation technologies including model-to-model with ATL and model-to-text with MOFScript. It is shown how method engineering can be supported by the OMG SPEM standard and the Eclipse EPF framework.

Part II focuses on *SOA - Service Oriented Architectures* – with a basis in concepts for service oriented computing, with a special emphasis on technologies for web services with XML, WSDL and BPEL. The basis technologies for the semantic web is also introduced with RDF and OWL, and semantic web services with OWL-S and WSMO. A last section presents agent-oriented computing with multi agent systems (MAS) and a platform independent model for agents (PIM4Agents).

Part III focuses on *MDE for SOA - Model Driven Engineering for Service Oriented Architectures* – and applies the principles of model driven engineering to service oriented architectures. The starting point for this approach is the COMET methodology used in previous INF5120 courses, this year enhanced to become COMET-S for Services through the use of new standard UML profiles and metamodels. The Business model uses in particular BMM (Business Motivation Metamodel, BPMN (Business Process Modeling Notation). The Requirements model supports mappings from use cases to services definitions. The service architecture model uses the new UPMS (UML Profile and Metamodel for Services). The platform specific model will vary depending on the target platform. The course has been using the JEE platform as reflected in the Oblig exercises in the course.

Part IV focuses on *MDI - Model Driven Interoperability* – and illustrates how a model driven approach can be applied to the problem domain of interoperability through the use of horizontal mappings and transformations. The approach to this is illustrated with the AIF (ATHENA Interoperability Framework) and the AIM (ATHENA Interoperability Methodology) and a set of articles on MDI.

I Model Driven Development

I.1 Introduction

The object technology revolution has allowed the replacement of the more than twenty-years old step-wise procedural refinement paradigm by the more fashionable object composition paradigm. Surprisingly this evolution seems itself today to be triggering another even more radical change, towards model-based technology. As a concrete trace of this, the Object Management Group (OMG) is rapidly moving from its previous Object Management Architecture vision (OMA) to the newest Model-Driven Architecture (MDA™) [OMG MDA] and even Microsoft is putting investment in its own variant of modelling around the notion of Domain Specific Modelling [Cook jan04].

The object technology revolution has allowed the replacement of the more than twenty-years old step-wise procedural refinement paradigm by the more fashionable object composition paradigm. Surprisingly this evolution seems itself today to be triggering another even more radical change, towards model-based technology. As a concrete trace of this, the Object Management Group (OMG) is rapidly moving from its previous Object Management Architecture vision (OMA) to the newest Model-Driven Architecture (MDA™) [OMG MDA] and even Microsoft is putting investment in its own variant of modelling around the notion of Domain Specific Modelling [Cook jan04].

MDA is the OMGs instantiation of an approach to software development coming to be known as Model Driven Engineering (MDE) or Model Driven Development (MDD). MDD focuses on Models as the primary artefacts in the development process, with Transformations as the primary operation on models, used to map information from one model to another. There is presently an important paradigm shift in the field of software engineering that may have important consequences on the way information systems are built and maintained. Presenting their "*software factor*y" approach, J. Greenfield and K. Short write in [Greefield, Short]:

> *"The software industry remains reliant on the craftsmanship of skilled individuals engaged in labor intensive manual tasks. However, growing pressure to reduce cost and time to market and to improve software quality may catalyze a transition to more automated methods. We look at how the software industry may be industrialized, and we describe technologies that might be used to support this vision. We suggest that the current software development paradigm, based on object orientation, may have reached the point of exhaustion, and we propose a model for its successor."*

The central idea of object composition is progressively being replaced by the notion of model transformation. One can view these in continuity or in rupture. The idea of software systems being composed of interconnected objects is not in opposition with the idea of the software life cycle being viewed as a chain of model transformations.

In November 2000, the OMG made public the MDA initiative, a particular variant of a new global trend called *model driven development*. The basic ideas of MDD are germane to many other approaches such as generative programming, domain specific languages, model-integrated computing, software factories, etc. MDA may be defined as the realization of MDD principles around a set of OMG standards like MOF, XMI, OCL, UML, CWM, SPEM, etc. MDD is presently making several promises about the potential benefits that could be reaped from a move from code-centric to model-based practices. When we observe these claims, we may wonder when they may be satisfied: on the short, medium or long term or even never perhaps for some of them.

The MDA approach does not have a unique goal but multiple goals. Among the objectives pursued, one may list the separation from business-neutral descriptions and platform dependent implementations, the identification, precise expression, separation and combination of specific aspects of a system under development with domain-specific languages, the establishment of precise relations between these different languages in a global framework and in particular the possibility to express operational transformations between them.

MDD is an evolving paradigm with many expectations on its final potential and with much research and development needed before it will meet those expectations. As stated by Steve Cook []:

> *"Over the past few years, new modelling technologies have begun to emerge … under the banner of Model Driven Architecture (MDA®) have created a buzz of interest by promising to increase the productivity of software development and the portability of software. On the other hand, we can see parallels between the promotion of MDA and the promotion of Computer-Aided Software Engineering (CASE) tools during the 1980s, and CASE clearly failed to live up to its promises. We should be very sceptical about new claims for model-driven software development unless we can demonstrate new ways of thinking about the problem that will avoid the pitfalls and failure of CASE."*

In general we can view Model Driven Development as a general principle for software engineering that can be realised in a number of different ways (using different standards) and supported by a variety of tools. The following table illustrates this, showing three current realisations of the MDD principle, with tools that are design in support of a particular standard.

Principle	Model Driven Development		
Standard	MDA	DSM	MIC
Supporting Tools	EMF	Visual Studio 2005	GME

Table 1 Three approaches to MDD

I.2 Principles of Modelling

The attraction of models is that they enable a problem to be precisely described without having to delve into the technical details; essentially a model is an abstract description.

> *"By model we mean a complex structure that represents a design artefact, such as a relational schema, object-oriented interface, UML model, XML DTD, web-site schema, semantic network, complex document, or software configuration. Many uses of models involve managing changes in models and transformations of data from one model into another. These uses require an explicit representation of mappings between models. We propose to make database systems easier to use for these applications by making model and model mapping first-class objects with special operations that simplify their use. We call this capacity model management."*
> *[P.A. Bernstein, A.L. Levy & R.A. Pottinger MSR-TR-2000-53]*

Modelling is a fundamental part of Model Driven Development and is in itself a far old discipline than computing. However, even in the context of computing systems, modelling is far older than the recent acknowledgement through MDD that it can form a major and useful part of system engineering. Modelling has been advocated as an important part of system design for almost as long

as we have been building computing systems. Early approaches to modelling computing systems such as Yordon Structured Design (late 1970s), DeMarco Structured Analysis (late 1970s), Merise (late 1970s), Finkelstein/Martin Information Engineering (1970/80) Structured System Analysis and Design Method (SSADM and LSDM) (early 1980s), Shlaer Mellor (early 1980s), Booch (early 1990s) were used to aid system building as long as 30 years ago.

Since then numerous other informal and formal approaches to modelling have been developed. We could not possible do a complete review of all the past and present approaches to modelling, however it is important to point out that:

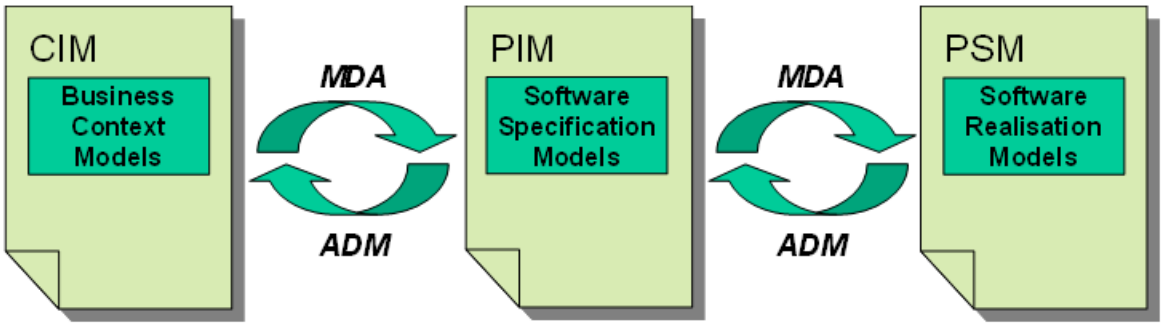
a) modelling is not in its self a new concept
b) modelling approaches existed before UML, MDD, MDA, MOF or the OMG
c) UML is not the only language for modelling
d) Modelling need not be Object-Oriented.

UML is currently one of the most widely used modelling languages. Despite the many objections to and problems with it, it remains accepted by industry and supported by numerous tool vendors.
Prior to UML perhaps some of the most widely adopted modelling languages were Entity Relationship Diagrams for modelling data, SDL for modelling telecommunication systems and perhaps languages such as Z, B, CSP, LOTOS etc. for more formal modelling of systems and processes.

I.3 Model Driven Architecture

I.3.1 Model-driven development (MDD)

Model-driven development (MDD) represents an approach to system engineering where models are used in the understanding, design, construction, deployment, operation, maintenance and modification of software systems. Model transformation tools and services are used to align the different models, ensuring that they are consistent across e.g. different refinement levels.



Model-driven development in our view represents a business-driven approach to software systems development that starts with a computation independent model (CIM) describing the business context and business requirements. The CIM is refined to a platform independent model (PIM) which specifies services and interfaces that the software systems must provide to the business, independent of software technology platforms. The PIM is further refined to a platform specific model (PSM) which describes the realisation of the software systems with respect to the chosen software technology platforms. In addition to the business-driven approach, a model-driven framework should also address how to integrate and modernise existing legacy systems according to new business needs. This approach is known as architecture-driven modernisation (ADM) in the OMG.

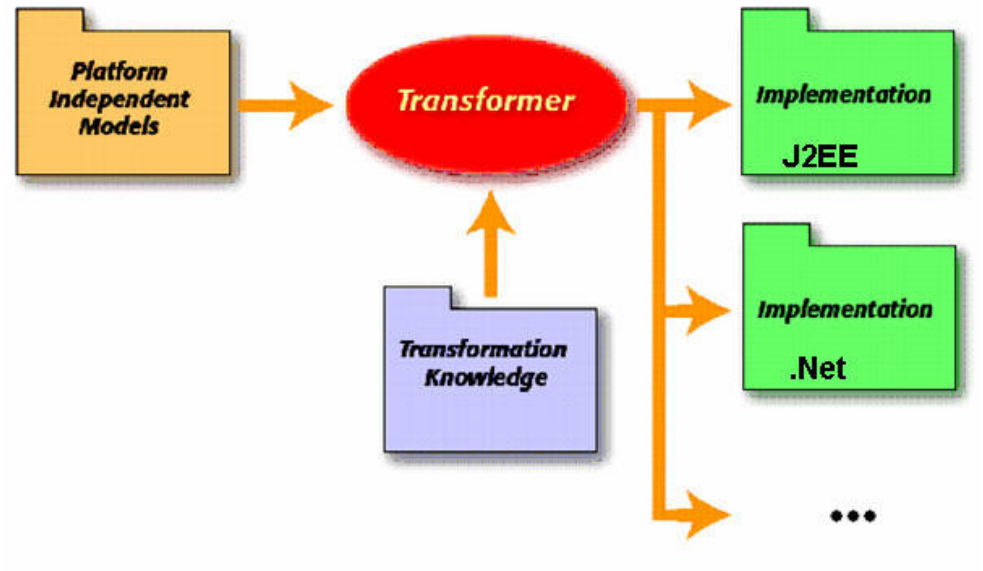
I.3.2 Model-driven architecture (MDA)

The current state of the art in Model-Driven Engineering (MDE) is much influenced by the ongoing standardisation activities around the OMG Model Driven Architecture® (MDA®).

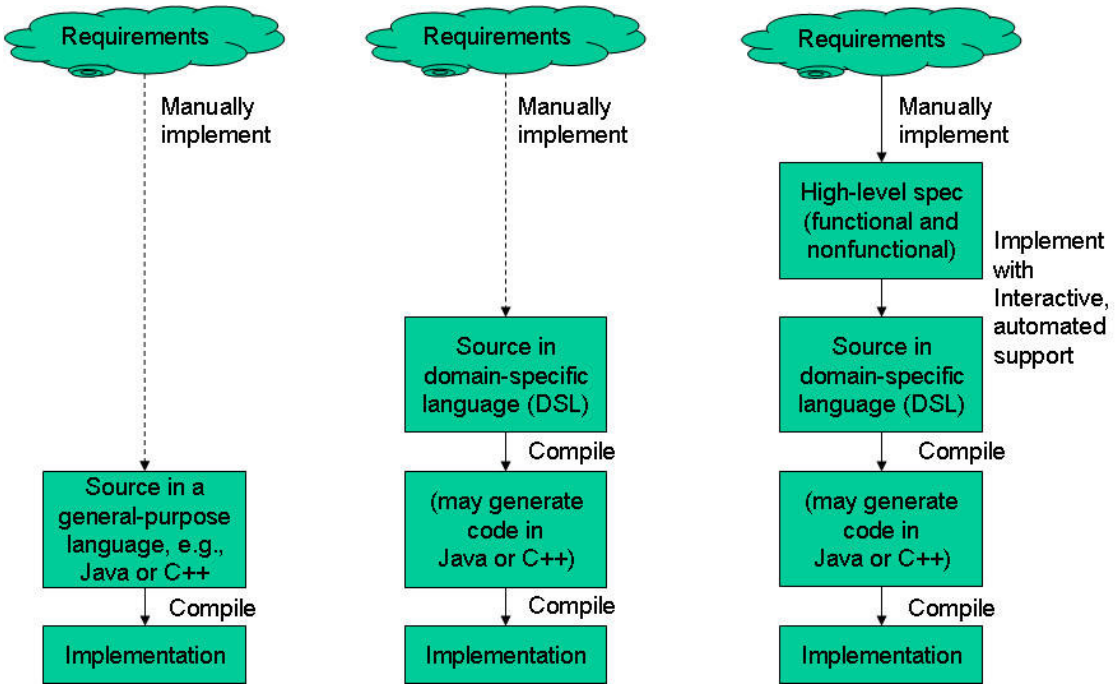
- MDA is a framework which defines a model-driven approach to software systems development.
- MDA encapsulates many important ideas - most notably the notion that real benefits can be obtained by using visual modelling languages to integrate the huge diversity of technologies used in the development of software systems.

I.3.2.1 MDA from 30.000 feet

MDA promotes the idea of designing software systems at a platform-independent model (PIM) level which can be transformed to software implementations with model transformation technologies that incorporates the knowledge of the execution platforms in question. A PIM can be retargeted to different platforms.



One of the original goals of model-driven development was to increase automation in software development. Bridging the gap between requirements and manual implementation is done by introducing new modelling and abstraction layers where development tools can provide interactive and automated support for software implementation.



I.3.2.2 Goals

The three primary goals of MDA are portability, interoperability and reusability. The MDA starts with the well-known and long established idea of separating the specification of the operation of the system from the details of the way the system uses the capabilities of its software execution platform (e.g. J2EE, CORBA, Microsoft .NET and Web services).

MDA provides an approach for:

- specifying a system independently of the software execution platform that supports it;
- specifying software execution platforms;
- choosing a particular software execution platform for the system;
- transforming the system specification into one for a particular software execution platform;

I.3.2.3 Basic concepts

The OMG MDA builds on the following basic concepts:

- **System:** Existing or planned system. System may include anything: a program, a single computer system, some combination of parts of different systems
- **Model:** A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text.
- **Architecture:** The architecture of a system is a specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors. MDA prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.
- **Viewpoint:** A viewpoint on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system.

- **View:** A viewpoint model or view of a system is a representation of that system from the perspective of a chosen viewpoint.
- **Platform:** A platform is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.

I.3.2.4 Model-driven development process

A system development process is model-driven if

- the development is mainly carried out using conceptual models at different levels of abstraction and using various viewpoints
- it distinguishes clearly between platform independent and platform specific models
- models play a fundamental role, not only in the initial development phase, but also in maintenance, reuse and further development
- models document the relations between various models, thereby providing a precise foundation for refinement as well as transformation

I.3.2.5 Three main abstraction levels

The OMG MDA defines three main abstraction levels:

- Computation independent model (CIM)
    - The computational independent viewpoint is focused on the environment of the system and on the specific requirements of the system.
    - A CIM represents the computational independent viewpoint.
    - The CIM hides the structural details and, of course, the details related to the targeted platform.
- Platform independent model (PIM)
    - A platform independent model is a view of the system from a platform independent viewpoint.
    - The platform independent viewpoint is focused on the operation of the system, hiding the platform specific details.
    - A PIM exhibits platform independence and is suitable for use with a number of different platforms of similar types.
    - The PIM gathers all the information needed to describe the behaviour of the system in a platform independent way.
- Platform specific model (PSM)
    - A platform specific model is a view of the system from the platform specific viewpoint.
    - A PSM combines the specifications in the PIM with the details that specify how the system uses a particular type of platform.
    - The PSM represents the PIM taking into account the specific platform characteristics.

I.4 MDD Standards

I.4.1 OMG's Model Driven Architecture (MDA)

As defined by the OMG (in initial MDA Guide draft), see [11] for latest MDA Guide.

"

> *The following was approved unanimously by 17 participants at the ORMSC plenary session, meeting in Montreal on 23 August 26, 2004. The stated purpose of these two paragraphs was to provide principles to be followed in the revision of the MDA Guide:*
> *MDA is an OMG initiative that proposes to define a set of non-proprietary standards that will specify interoperable technologies with which to realize model-driven development with automated transformations. Not all of these technologies will directly concern the transformations involved in MDA. MDA does not necessarily rely on the UML, but, as a specialized kind of MDD (Model Driven Development), MDA necessarily involves the use of model(s) in development, which entails that at least one modeling language must be used. Any modeling language used in MDA must be described in terms of the MOF language, to enable the metadata to be understood in a standard manner, which is a precondition for any ability to perform automated transformations."*

The three primary goals of MDA are portability, interoperability and reusability. Over the last dozen years, the Object Management Group, better known as OMG, standardized the object request broker (ORB) and a suite of object services. This work was guided by the Object Management Architecture (OMA), which provides a framework for distributed systems and by the Common ORB Architecture, or CORBA™, a part of that framework.

The OMA and CORBA were specified as a software framework, to guide the development of technologies for OMG adoption. This framework is in the same spirit as the OSI Reference Model and the Reference Model of Open Distributed Processing (RM-ODP or ODP [ISO/IEC 10746-1]). The OMA framework identifies types of parts that are combined to make up a distributed system and, together with CORBA, specifies the types of connectors and the rules for their use.
Starting in 1995, OMG informally began to adopt industry-specific ("domain") technology specifications. Recognizing the need to formalize this activity, OMG added the new Domain Technology Committee in the major process restructuring of 1996 and 1997.
Parallel work around object modelling, resulted in the adoption of the Unified Modelling Language, UML. OMG members then began to use UML, sometimes in replacement for IDL, in the specification of technologies for OMG adoption.

In keeping with its expanding focus, OMG began the development of a second framework, the Model Driven Architecture™ or MDA [OMG MDA].
MDA is not, like the OMA and CORBA, a framework for implementing distributed systems. It is an approach to using models in software development. The Model Driven Architecture starts with the well-known and long established idea of separating the specification of the operation of the system from the details of the way the system uses the capabilities of its platform. MDA provides an approach and tools for:

- specifying a system independently of the platform that supports it,
- specifying platforms
- choosing a particular platform for the system, and
- transforming the system specification into one for a particular platform.

The MDA core is based basically on the following standards: MOF (Meta Object Facility), UML (Unified Modelling Language) and CWM (Common Warehouse Metamodel). There are also a series of Model Interchange standards based around XML (XMI) Java (JMI) and CORBA (CMI).

- **MOF** plays an important role because in this level we can define transformations between models and we can also define others metamodels.
- **UML** is a unified modelling language and it is not a universal modelling language. Moreover we need to specify that UML is not a methodology. Optionnally a methodology could be defined within the Model Driven Engineering (MDE). In addition UML could be tailored for a specific purpose and domain through constraints and extensions mechanisms. This kind of UML is called UML Profile and there are some UML Profiles under standardization process and others Profiles that are already standardized (e.g. UML Profile for Enterprise Application Integration, UML Profile for Enterprise Distributed Object Computing,…). In the above pictures, the UML Profile concept is not include because it is considered as an UML ability.
- **CWM** is a language metamodel to specify the design and use of the data warehouse.
- **XMI** is a mapping to the XML technical space, mainly intended to allow models and metamodels to be exchanged after serialization
- **JMI** is a mapping to the Java technical space, in order to allow access to the internal structure of models and metamodels by executable Java programs
- **CMI** is a mapping to the CORBA technical space, in order to show the interoperabilty of main OMG standards

There are other definitions of MDA; of particular note is that included in An MDA Manifesto [MDA Manifesto] as published by the MDA Journal:
"In essence, the foundations of MDA consist of three complementary ideas:

1. **Direct representation**. Shift the focus of software development away from the technology domain toward the ideas and concepts of the problem domain. Reducing the semantic distance between problem domain and representation allows a more direct coupling of solutions to problems, leading to more accurate designs and increased productivity.
2. **Automation**. Use computer-based tools to mechanize those facets of software development that do not depend on human ingenuity. One of the primary purposes of automation in MDA is to bridge the semantic gap between domain concepts and implementation technology by explicitly modeling both domain and technology choices in frameworks and then exploiting the knowledge built into a particular application framework.
3. **Open standards**. Standards have been one of the most effective boosters of progress throughout the history of technology. Industry standards not only help eliminate gratuitous diversity but they also encourage an ecosystem of vendors producing tools for general purposes as well as all kinds of specialized niches, greatly increasing the attractiveness of the whole endeavor to users. Open source development ensures that standards are implemented consistently and encourages the adoption of standards by vendors."

Another primary source of MDA definitions is the OMG's MDA™ Guide [OMG MDA]. This document defines the main set of concepts related to Model Driven Architecture, including definitions of CIM, PIM and PSM. Finally, there are many books starting to appear that all introduce MDA with the authors giving their own flavour to the definition [Frankel] [Kleppe, Warmer, Bast] [Mellor etal].
In revised versions of the OMG's MDA™ Guide [OMG MDA] *the model, view and viewpoint principles of architecture for systems is emphasised. It is made clear that a system can be many things, i.e. both an IT system, and an Enterprise, or a Federation of enterprises. Many viewpoints of a system can exist, and the defined terms CIM, PIM and PSM are just examples. Typically*

refinements and other relationships between models in different views can be done in multiple levels, and there is no absolute CIM-PIM-PSM triad. (Added in version 2.0 of this report).

I.4.2 Microsoft's Domain Specific Modelling (DSM)

There is so far little published material regarding Microsoft's approach to MDD, perhaps one of the best sources of information is the ongoing debate within a series of articles published by the MDA Journal [MDA Journal] and on the Web site mentioned by Steve Cook in his first article that gives details of Microsoft's modelling tools. The place to watch for upcoming info is http://lab.msdn.microsoft.com/vs2005/teamsystem/workshop/.

Microsoft released in October 2004 a preview version of new tools intended to make it easier for companies to create custom Web applications. This was released as a "community technology preview" version of modelling tools, formerly code-named Whitehorse, to be included in Visual Studio 2005 Team System, an upcoming addition to Microsoft's line of developer packages that focuses on enterprise developers.

Domain-specific language tools lay the foundation for software factories by providing a framework and a set of tools for delivering domain-specific visual designers that plug into Visual Studio Team System. These designers could be tools for industry verticals, such as the Financial/ERP, health care or telecommunications industries or, they could be tools for development across numerous disciplines, such as object-oriented modeling and architecture.

Other vendors, such as IBM and Borland Software, also have invested substantially in modelling. Borland announced its own modelling tools, called Together Architect, and IBM has just released the new Rational Software Architect . High-quality software that doesn't easily crash or require frequent maintenance is especially important for a company's most significant applications. Market researcher Gartner estimates that the average cost of unplanned downtime for so-called "mission critical" software is $100,000 per hour. Fully 40 percent of application failures are due to software problems, according to Gartner.

One of the most immediate concerns development tool companies have is preparing corporate customers for building new software following a service-oriented architecture (SOA), with a focus on n more flexible, better quality software at a lower cost. An SOA, for example, could allow an e-commerce site to perform a complex transaction involving different business partners by linking together several Web services, rather than requiring programmers to hand-code connections to partners.

By publishing the software development kit for the modelling tools in Visual Studio, Microsoft hopes to encourage partners and customers to create customised models components to describe software functions peculiar to specific industries and tasks. The new Visual Studio versions will be one of the first steps in a big Microsoft effort, dubbed "Software Factories", to enable companies to produce customised applications faster by automating routine tasks.

Developers who are using UML, data modeling or business process modeling tools are often faced with the need to personalize or customize these tools. They turn them into domain-specific, company-specific and even project-specific modeling tools. They modify what a diagram shows and what it means, and they add code generators, report generators, and functionality to do useful things with their models. Often they will write custom code to import and export their models. In some cases, they even devise their own interpretation of standard languages like UML. But in other cases, a developer may simply be reusing a pattern of code with a template language or XML schema for which he writes a code generator for specific projects. Developers who are interested in using models and generators like this are indulging in model-driven development as a way to

achieve higher productivity. The 'Domain-Specific Language (DSL) Concepts Editor' will enable developers, both tool builders and tool users, to create their own custom and problem-specific modeling tools with little effort.

The intention is to provide tools to build graphical designers that are focused on specific aspects of application building. However, this does not necessarily mean a need to invent entirely new notations. One can use the same well-known notations for statecharts etc. in these custom tools, but one will have the freedom to extend or redefine the notation, and describe precisely what the notation means in the problem domain. The 'DSL Concepts Editor' also enables to define concept models that are not necessarily used in graphical designers or modeling tools.

I.4.3 Model Integrated Computing (MIC)

Model-Integrated Computing (MIC) addresses the problems of developing software integrated systems by providing rich, domain-specific modeling environments including model analysis and model-based program synthesis tools. This technology is used to create and evolve integrated, multiple-aspect models using concepts, relations, and model composition principles routinely used in the specific field, to facilitate systems/software engineering analysis of the models, and to automatically synthesize applications from the models. MIC has been used to develop many different technologies and solutions for industry and government.
Model-Integrated Computing (MIC) has been developed over a decade at ISIS, Vanderbilt University for building embedded software systems. The key element of this approach is the extension of the scope and usage of models such that they form the "backbone" of a model-integrated system development process. See http://www.isis.vanderbilt.edu/research/mic.html

In Model-Integrated Computing models play the following central roles:

- Integrated, multiple-view models capture the information relevant to the system to be developed. Models can explicitly represent the designer's understanding of the entire system, including the information processing architecture, the physical architecture, and the environment it operates in.
- Integrated modeling allows the explicit representation of dependencies and constraints among the different modeling views.
- Tools analyze different, but interdependent characteristics of systems (such as performance, safety, reliability, etc.). Tool-specific model interpreters translate the information in the models to the input languages of analysis tools.
- The integrated models are used to automatically synthesize the software. The model-integrated program synthesis process utilizes model interpreters to translate the models into executable specifications.
- UML-based metaprogramming interface allows synthesis, evolution of domain-specific MIPS environments.

Using MIC technology one can capture the requirements, actual architecture, and the environment of a system in the form of high-level models. The requirement models allow the explicit representation of desired functionalities and/or non-functional properties. The architecture models represent the actual structure of the system to be built, while the environment models capture what the "outside world" of the system looks like. These models act as a repository of information that is needed for analyzing and generating the system.

Multigraph Architecture

The MultiGraph Architecture (MGA) provides a unified software architecture and framework for building domain-specific tools for: (1) building, testing, and storing domain models, (2) transforming the models into executable programs and/or extracting information for system engineering tools, and (3) integrating applications on heterogeneous parallel/distributed computing platforms. The MGA is comprised of three levels as described below.

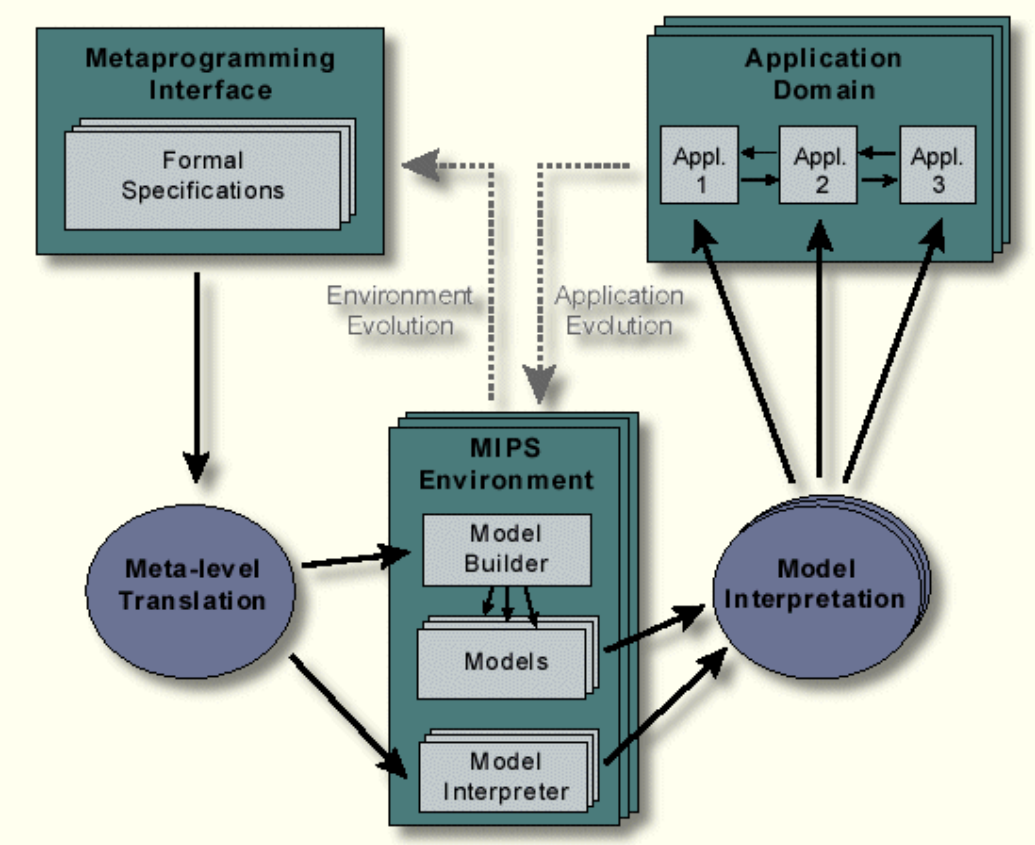


Figure 1 Multigraph Architecure

Application Level

The Application Level represents the synthesized, adaptable software applications. The executable programs are specified in terms of the Multigraph Computational Model (MCM). The MCM is a macro-dataflow model which models the synthesized programs as an attributed, directed, bipartite graph. The runtime support for MCM, the Multigraph Kernel, is implemented as an overlay above operating and communication systems.

Model-Integrated Program Synthesis (MIPS) Level

The Model-Integrated Program Synthesis (MIPS) Level includes generic, but customizable tools for model building, model analysis, and application synthesis. The generic components of the architecture are: (1) a customizable Graphical Model Editor (GME), (2) a database layer for storing and accessing models, (3) model analysis tools and external analysis tools, and (4) model interpreters that synthesize applications (executable models) in terms of the MCM, or translate models into input data structures of the analysis tools (analysis models).

The Meta-Level

The Meta-Level of MGA provides a metaprogramming interface for the components on the MIPS Level. The Metaprogramming Interface includes: (a) support for the formal specification of domainspecific modeling paradigms and model interpreters using formal languages, (b) metalevel translators to generate configuration files for the GME from the modeling paradigm specification, (c) metalevel program synthesis tools for generating model interpreters from their formal specification, and (d) support for the validation and verification of the metamodels. Metamodels capture the formal semantics of domain specific modeling languages and model interpreters. The formal semantics of modeling paradigms define the constraints that the domain models must satisfy with respect to the concepts, relations, model composition principles and domain-specific integrity

constraints. As we can consider applications as "executable instances" of domain models, the domain models can be viewed as "instances" of metamodels.

Impact

- metaprogramming tools significantly decrease the required effort to create integrated domain specific modeling environments,
- metaprogramming tools decrease the development time of generators,
- MIPS environments enable the rapid modification/adaptation of applications by simply modifying domain specific models, and
- metaprogramming toolset supports environment evolution (i.e., changing the modeling paradigm).

I.5 Metamodelling

I.5.1 What is a metamodel ?

Metamodelling is a controversial topic which is currently critical within the UML/OMG/MDA community.

- A metamodel is just another model (e.g. written in UML) and is thus a model of a set of models.
- Metamodels are specifications. Models are valid if no false statements according to metamodel (e.g. well-formed)
- Metamodels typically represents domain-specific models (real-time systems, safety critical systems, e-business)
- The domain of metamodelling is language definition. A metamodel is a model of some part of a language. Which part depends on how the metamodel is to be used. Examples of parts are syntax, semantics, views and diagrams.

Then we also have what is known as a meta-metamodel. This can said to be a model of metamodels. Reflexive metamodel are expressed using itself. A minimal reflexive metamodel contains all of the parts used to describe the set of models that are of interest.

In its broadest sense, a metamodel is a model of a modelling language. The term ”meta” means transcending or above, emphasising the fact that a metamodel describes a modelling language at a higher level of abstraction than the modelling language itself. In order to understand what a metamodel is, it is useful to understand the difference between a metamodel and a model. Whilst a metamodel is also a model, a metamodel has two main distinguishing characteristics.

1. Firstly, it must capture the essential features and properties of the language that is being modelled. Thus, a metamodel should be capable of describing a language’s concrete syntax, abstract syntax and semantics.
2. Secondly, a metamodel must be part of a metamodel architecture. Just as we can use metamodels to describe the valid models or programs permitted by a language, a metamodel architecture enables a metamodel to be viewed as a model, which itself is described by another metamodel. This allows all metamodels to be described by a single metamodel. This single metamodel, sometimes known as a meta-metamodel, is the key to metamodelling as it enables all modelling languages to be described in a unified way.

I.5.2 Why metamodel?

System development is fundamentally based on the use of languages to capture and relate different aspects of the problem domain. The benefit of metamodelling is its ability to describe these languages in a unified way. This means that the languages can be uniformly managed and manipulated thus tackling the problem of language diversity. For instance, mappings can be constructed between any number of languages provided that they are described in the same metamodelling language. Another benefit is the ability to define semantically rich languages that abstract from implementation specific technologies and focus on the problem domain at hand. Using metamodels, many different abstractions can be defined and combined to create new languages that are specifically tailored for a particular application domain. Productivity is greatly improved as a result.

Uses for a metamodel can be summarised as follows:

- Define the syntax and semantics of a language.
- Explain the language.
- Compare languages rigorously.
- Specify requirements for a tool for the language.
- Specify a language to be used in a meta-tool.
- Enable interchange between tools.
- Enable mapping between models.

I.5.3 How to metamodel?

There is a clearly defined process to constructing metamodels, which does at least make the task a well-defined, if iterative, process. The process has the following basic steps:

1. defining abstract syntax
2. defining well-formedness rules and meta-operations
3. defining concrete syntax
4. defining semantics
5. constructing mappings to other languages

I.5.3.1 Abstract syntax

The metamodel describes the abstract syntax of a language. The abstract syntax of a language describes the vocabulary of concepts provided by the language and how they may be combined to create models. It consists of a definition of the concepts, the relationships that exist between concepts and well-formedness rules that state how the concepts may be legally combined.

I.5.3.2 Concrete syntax

I.5.3.2.1 Visual syntax

A visual syntax presents a model or program in a diagrammatical form. A visual syntax consists of a number of graphical icons that represent views on an underlying model.

I.5.3.2.2 Textual syntax

A textual syntax enables models or programs to be described in a structured textual form. A textual syntax can take many forms, but typically consists of a mixture of declarations, which declare specific objects and variables to be available, and expressions, which state properties relating to the declared objects and variables.

I.5.3.3 Semantics

An abstract syntax conveys little information about what the concepts in a language actually mean. Therefore, additional information is needed in order to capture the semantics of a language. Defining a semantics for a language is important in order to be clear about what the language represents and means.

I.5.4 Metamodeling standards and technologies

I.5.5 OMG MDA standards

In order to support the MDA approach to software development we need standards to support the description of enterprise and domain models, as well as technical models at the platform-independent and platform-specific levels. Furthermore we need standards for the management and transformation of such models.

Fortunately OMG provides a set of specifications that are publicly available at http://www.omg.org/mda/specs.htm.

I.5.5.1 Unified Modeling Language (UML) and profiles

UML 2.0 is the de-facto standard industry language for specifying and designing software systems. UML addresses the modelling of architecture and design aspects of software systems by providing language constructs for describing, software components, objects, data, interfaces, interactions, activities etc. UML now provides support for a wide variety of modelling domains, including real-time system modelling and is used more and more in embedded systems.

The realization of DSL's through profiling is based on the existing UML2.0 standard. The OMG(Object Management Group) defines the UML profiles as a way to tailor the languages to specific areas. This areas can be from business modeling, to enterprise architectures, service oriented architectures, QoS modeling or many other technologies.

Standard profiles developed by OMG are listed at Modeling Standards Page.

The advantage with UML profiles is the existing UML standard and the tool support for it. This reduces the effort in learning a new language and developing new tools. The tools that supports UML profiling are both open source and commercial. Some of the most used in the open sourse world are Eclipse or ArgoUML , while other proprietary tools are Borland Together 2006 for Eclipse , IBM Rational Software Architect and Modeler or Microsoft Visio.

A list of OMG's UML2 tools is provided at: UML2.0 tools

While UML profiles are quiet supported in the marked and the open source community, developing DSL's from scratch seems to be a more tricky task. The mainstream technologies for modeling DSL's are divided in those which use standardized MDA facilities and those based on the Software Factory idea.

I.5.5.2 Meta Object Facility (MOF)

MOF 2.0 provides the standard modelling and interchange constructs that are used in MDA. These constructs are a subset of the UML modelling constructs. This common foundation

provides the basis for model/metadata interchange and interoperability. MOF plays an important role in defining transformations between models.

I.5.5.3 XML Metadata Interchange (XMI)

XMI 2.1 is a format to represent models in a structured text form. In this way UML models and MOF metamodels may be interchanged between different modelling tools.

I.5.5.4 MOF Queries/View/Transformations (QVT)

QVT 2.0 provides a standard specification of a language suitable for querying and transforming models which are represented according to a MOF metamodel.

I.5.5.5 Software Process Engineering Metamodel (SPEM)

SPEM 1.1 provides a standard for representing software development methods.

I.5.5.6 Common Warehouse Metamodel (CWM)

CWM 1.1 is the OMG data warehouse standard. It covers the full life cycle of designing, building and managing data warehouse applications and supports management of the life cycle.

I.5.6 Eclipse MDA technologies

The Eclipse community has started to provide technology support for the OMG model-driven architecture (MDA).

I.5.6.1 Eclipse Modeling Framework (EMF)

http://www.eclipse.org/emf/

"EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Models can be specified using annotated Java, XML documents, or modeling tools like Rational Rose, then imported into EMF. Most important of all, EMF provides the foundation for interoperability with other EMF-based tools and applications."

We can say that EMF provides a low cost entry, because an EMF model requires just a small subset of the kinds of things that you can model in UML, specifically simple definitions of the classes and their attributes and relations, for which a full-scale graphical modeling tool is unnecessary.

Actually, EMF started out as an implementation of the MOF specification but evolved from there based on the experience we gained from implementing a large set of tools using it. EMF can be thought of as a highly efficient Java implementation of a core subset of the MOF API. However, to avoid any confusion, the MOF-like core meta model in EMF is called Ecore.

The Eclipse Modeling Framework (EMF) is a Java framework and code generation facility for building tools and other applications based on a structured model. Eclipse an open source software development project led by IBM. Eclipse intents to provide a kind of universal tool platform – an open extensible IDE for anything and nothing in particular [Eclipse]. EMF is one of subproject of Eclipse.

EMF can be thought of as MDA on training wheels. EMF uses XMI (XML Metadata Interchange) as its canonical form of a model definition. And EMF supports the implementation of OMG (Object Management Group) MOF (Meta Object Facility) specification. EMF is a highly efficient Java implementation of a core subset of the MOF API [EMF].

An EMF model is just a small subset of UML model, especially simple definitions of the classes and their attributes and relations. An EMF model can be defined under Eclipse environment by:

- Importing from a Rational Rose class model
- Describing the model directly in an XMI document
- Defining the model using annotated Java
- Using XML Schema to describe the form of a serialization of the model

When an EMF model is specified, the EMF generator can create a corresponding set of Java implementation classes. These generated classes can be edited by adding methods and instance variables and still regenerated from the model as needed: the additions will be preserved during the regeneration. If the code you added depends on something that is changed in the model, your code has to be updated to reflect those changes; otherwise, your code is completely unaffected by model changes and regeneration.
EMF consists of three fundamental frameworks: the core framework and EMF.Edit. The core framework provides basic generation and runtime support to create Java implementation classes for a model. EMF.Edit extends and builds on the core framework, adding support for generating adapter classes that enable viewing and command-based (undoable) editing of a model, and even a basic working model editor. EMF.Codegen provides code generation support.

I.5.6.1.1 The core EMF framework

The core EMF framework includes a meta model (Ecore) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically [What is EMF].
Ecore is the MOF-like core meta model in EMF. In the current proposal for MOF 2.0, a similar subset of the MOF model, which it calls EMOF (Essential MOF), is separated out. There are small, mostly naming differences between Ecore and EMOF; however, EMF can transparently read and write serializations of EMOF [EMF].

I.5.6.1.2 EMF.Edit

The EMF.Edit framework includes generic reusable classes for building editors for EMF models. It provides [EMF.Edit]:

- Content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop (JFace) viewers and property sheets.
- A command framework, including a set of generic command implementation classes for building editors that support fully automatic undo and redo.

I.5.6.1.3 EMF.Codegen

The EMF code generation facility is capable of generating everything needed to build a complete editor plug-in for the EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. The generation facility leverages the JDT (Java Development Tooling) component of Eclipse [What is EMF].

EMF provides a product quality, open source, model-driven tool for metadata-based tools integration in Eclipse [Gardener dec03]. With EMF, modelling and programming can be considered the same thing. Instead of forcing a separation of the high-level engineering/modelling work from

the low-level implementation programming, it brings them together as two well-integrated parts of the same job. Often, especially with large applications, this kind of separation is still desirable, but with EMF the degree to which it is done is entirely up to you [Budinsky]. EMF is widely used in IBM products for a variety of metamodels.

I.5.6.2 Eclipse Graphical Editing Framework (GEF)

http://www.eclipse.org/gef/

"The Graphical Editing Framework (GEF) allows developers to take an existing application model and quickly create a rich graphical editor."

I.5.6.3 Eclipse Graphical Modeling Framework (GMF)

http://www.eclipse.org/gmf/

"The Eclipse Graphical Modeling Framework (GMF) provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF. The project aims to provide these components, in addition to exemplary tools for select domain models which illustrate its capabilities."

GMF uses and therefore depends on EMF. GMF is not in anyway a replacement for EMF. EMF provides core facilities for defining models, and generating java code for manipulating and persisting instances of those models. The generated code includes an implementation of the observer pattern.

Because it is common for GEF editors to manipulate an object model (and use observer pattern) it pairs very well with EMF and has become a popular combination (see IBM redbook on using emf and gef).

The GMF project aims to simplify the combination of these two technologies by allows GEF editors to be specified and generated using models. i.e. GMF allows you to generate GEF editors for manipulating your EMF models.

I.5.6.4 Atlas Transformation Language (ATL)

http://www.eclipse.org/gmt/

The goal of the Eclipse Generative Modeling Tools (GMT) project is to produce a set of prototypes in the area of model-driven engineering (MDE). Amongst the technologies under the GMT project we find the Atlas Transformation Language (ATL) http://www.eclipse.org/gmt/atl/.

"The ATL project aims at providing a set of transformation tools for GMT. These include some sample ATL transformations, an ATL transformation engine, and an IDE for ATL (ADT: ATL Development Tools)."

I.5.6.5 Eclipse Process Framework (EPF)

http://www.eclipse.org/epf/

"The Process Framework Project has two goals:

- *To provide an extensible framework and exemplary tools for software process engineering - method and process authoring, library management, configuring and publishing a process.*
- *To provide exemplary and extensible process content for a range of software development and management processes supporting iterative, agile, and incremental development, and applicable to a broad set of development platforms and applications."*

I.6 UML profiles and DSLs

I.6.1 UML profiles

UML profiles allow us to adapt the UML language to the needs of the analysts or the application domain. Allows designers to model using application domain concepts. There are three extension mechanisms:

- Stereotypes
- Restrictions
- Tagged values

I.6.1.1 Stereotypes

A stereotype extends the vocabulary of UML with new construction elements derived from existing UML but specific to a problem domain. Can have associated restrictions and tagged values. Possibility of assigning an icon for a better graphical representation.

I.6.1.2 Restrictions

A restriction is a semantical condition represented by a textual expression. It imposes some kind of condition or requisite on the element to which it is applied. Restrictions are described in the Object Constraint Language (OCL).

I.6.1.3 Tagged values

A tagged value is a property associated to a model element. It is used to store information about the element such as management information, documentation, coding parameters, etc. Generally, the tools store this information but it is not shown in the diagrams.

I.6.1.3.1 How do we create DSL's?

DSL's can be realized in two ways, either by customization of pre-existing languages through profiles or by creating a new language with a standardized meta-data architecture based on Meta Object Facility (Meta Object Facility).

The first approach through customization is achieved by marking up UML concepts with existing stereotypes and tags. For example in PIM4SOA a "Class" is stereotyped to be an "Entity".

The second approach is based on the idea to create a brand new DSL from scratch. This involves using MDA facilities and standards to create a model of the DSL which is used to generate a tool for the it on an existing platform.

I.6.2 Domain-specific languages (DSLs)

DSL (Domain Specific Languages) are modelling languages designed for a specific purpose inside a problem domain. These languages provide abstractions that are tailored to a specific domain and their advantage is their ability to directly represent domain abstractions that fit the problem domain.

A DSL can be a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain. For a more specific definition of DSLs check out DSL: An Annotated Bibliography

An important aspect of DSL's is understanding the domain concept. In the software engineering world, a domain defines a set of common requirements, terminology, and concepts that is specific to a particular application domain. A DSL can therefore be any language that provides means of expressing these concepts and is created specifically to solve problems in this particular domain. In difference to general purpose-languages like UML (Unified Modeling Language) , a DSL is able to focus on concepts of the domain model and is not intended to be able to solve problems outside of it.

The main benefit of using a DSL is their ability to provide abstractions that are tailored to a specific problem domain and thereby a potential increase in productivity and ease of use. While it is also possible to abstract the concepts of the problem domain in UML, it is obviously more intuitive and efficient to be able to directly capture the specific domain abstractions used by the language. Some other benefits of DSL's are the possibility to raise the level of abstraction and the ability to produce a more precise model , since it is focused in a narrower view of the problem. This leads to a more flexible and agile product.

I.6.3 UML profiles vs. DSLs

While the OMG MDA promotes UML as the visual "universal" glue suitable for modelling everything, we are also seeing a trend towards development and co-existence of several domain-specific modelling languages, e.g. supported by the Microsoft Domain-Specific Language (DSL) tools (see http://msdn.microsoft.com/vstudio/DSLTools/)

Such approaches are now also being discussed in various OMG forums. UML is seen as a "general-purpose" language while DSLs may be more expressive for most purposes. DSLs (Domain Specific Languages) are modelling languages designed for a specific purpose inside a problem domain. These languages provide abstractions that are tailored to a specific domain and their advantage is their ability to directly represent domain abstractions that fit the problem domain.

A DSL can be a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain. An important aspect of DSL's is understanding the domain concept. In the software engineering world, a domain defines a set of common requirements, terminology, and concepts that is specific to a particular application domain. A DSL can therefore be any language that provides means of expressing these concepts and is created specifically to solve problems in this particular domain.

While the MDA approach treats UML, with customization, as the modelling language of choice for most application modelling, it also acknowledges the value of custom languages in certain specialized circumstances. This is the purpose of the OMG Meta-Object Facility (MOF) standard that plays an important role in MDD. UML itself is defined using MOF and there are MOF definitions of many other languages. Thus the MDA approach also allows the creation of DSLs using MOF as a basis rather than using UML profiles.

I.6.3.1 Advantages of using UML profiles

- UML is an open standard modelling language for which there are many available books and training courses.
- UML profiles provide a lightweight approach that is easily implemented using readily available UML tooling.
- Models with UML profiles applied can be read by all UML tools even if they do not have any knowledge of the profile.
- Basing all DSLs on UML creates a set of related languages that share common concepts.
- UML can be used for high-level architectural models as well as detailed models from which code can be generated.

I.6.3.2 Disadvantages of using UML profiles

- UML profiles only permit a limited amount of customization.
    - It is not possible to introduce new modelling concepts that cannot be expressed by extending existing UML elements.
- The use of UML does require familiarity with modelling concepts.

I.6.4 Other technologies

I.6.4.1 XMF-Mosaic

XMF-Mosaic is a model-based development platform based on Eclipse(open source IDE) and XMF . XMF stands for (eXecutable Metamodelling Facility) and is an extension of the existing standards such as MOF, QVT or OCL, with executable metamodelling capabilities This metamodelling facility provides the ability to create languages with all the key features(here semantics and syntax), the ability to create metamodels which involves having a meta-architecture(self defined languages)and the platform independency of metamodels, key "issue" of MDA.

I.6.4.2 Visual Studio 2005 DSL Tools

Visual Studio 2005 DSL Tools are a SDK(Standard Development Kit) that allows developers to use Microsoft's platform to build visual modelling tools that run inside of VisualStudio. DSL Tools are always associated to Software Factories. This is not rare since they realize ideas from Software Factories, but they should not be mistaken to be the Software Factories. DSLs are the first piece, they are the first product launched in the marked and more to come is predicted in the later versions of VisualStudio. The purpose of DSL Tools is to construct custom designed tools that can be used to model a problem domain. The first step is to create the model with the domain-specific notation. Then a running designer will be generated. This designer is based on the domain model created and a XML description for the user interface.

I.7 Model transformations

I.7.1 Model mapping and transformation

An important aspect of Model Driven Development (MDD) is model transformations, which allows automatically transformations of models. A model transformation is a transformation of

one or more source models to one or more target models, based on the meta models of each of these models. In other words the instances of one meta model is transformed into instances of another meta model.

Such transformations are defined by mapping rules. Each mapping rule describes what one, or more elements in the source model should be transformed to in the target model. When all mapping rules are applied, the mapping describes the complete transformation from the source model to the target model.

Thus given a source model, and the metamodels of both the source and the target models, one can automatically generate the target model by applying the correct mapping to the source model.

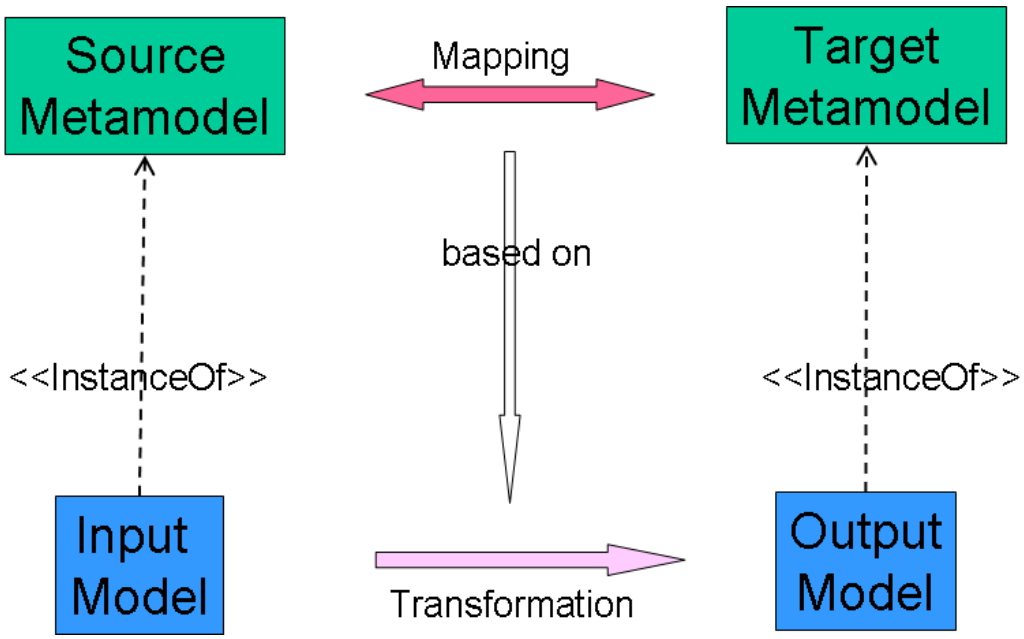
A common use of transformations is the transformations of Platform Independent Model (PIM) to Platform Specific Model(PSM), and PSM to code. The PIM should, as the name implies describe the system in a total platform independent way. Thus, whether your system stores data in a database or not, or is implemented in C, C++ or Java is in no interest here. The PIM simply captures what your system does, not how. After making a PIM of the system, one should then make PSM's for the different types of technologies used. The PSM should describe how the system is implemented, using a specific technology. In other words, if your system is implemented using Java, WSDL, and BPEL, you make one for each technology. When all the PSM's are made, one should create mapping rules describing the transformation from the PIM, to each of the PSM's.

As a last step, it's now possible to describe mappings from each of the PSM's to code. This code could be complete code, but most times it would be more of a skeleton of the code, where parts of the code would have to be manually entered.

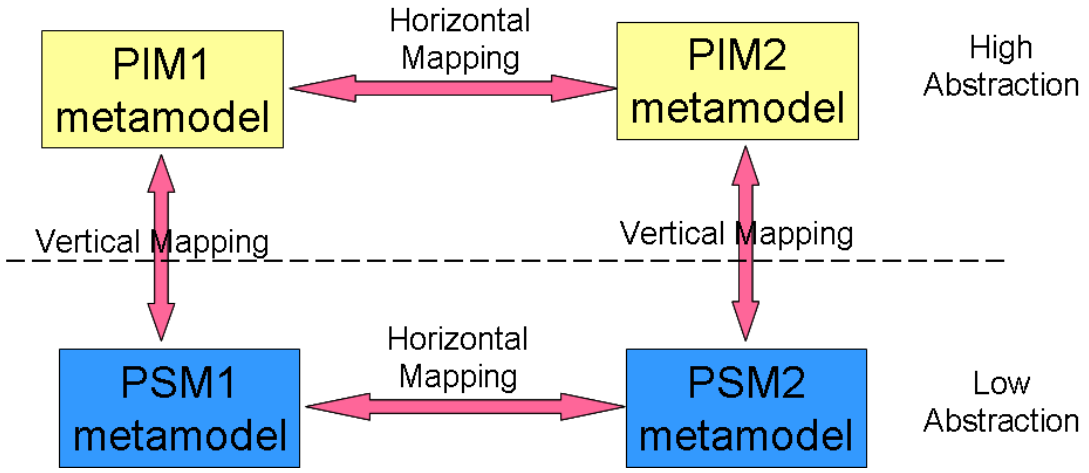
I.7.1.1 Mapping

Mapping is performed by defining relations between two models. The relations can be 1-to-1, n-to-1, 1-to-n or n-to-n. Mapping is performed in "design time".

The mapping is defined with models one meta-level higher than the input and output of the transformation. The mapping is used to perform transformation of instances of the mapped models. "The mapping describes the rules used for the transformation".



We can map between models that are on the “same” abstraction level illustrated with horizontal mapping, or we can map between abstraction levels illustrated with vertical mapping in the figure below.



I.7.1.2 Transformations

A transformations occur at "run-time" and takes input and produces output. A transformation is a one-way process which transforms according to a predefined mapping. Two main categories of transformation:

- Vertical transformation
- Horizontal transformation

In a vertical transformation the source model has the same level of abstraction as target model. Not to be confused with “meta-levels”. Examples of horizontal transformation:

- Refactoring
- Merging

In a horizontal transformation the source model is at a different level of abstraction than the target model. Examples of vertical transformation:

- Refinement (specialization)
  - PIM->PSM transformations
- Abstraction (generalization)

I.7.2 MDA standards for transformations

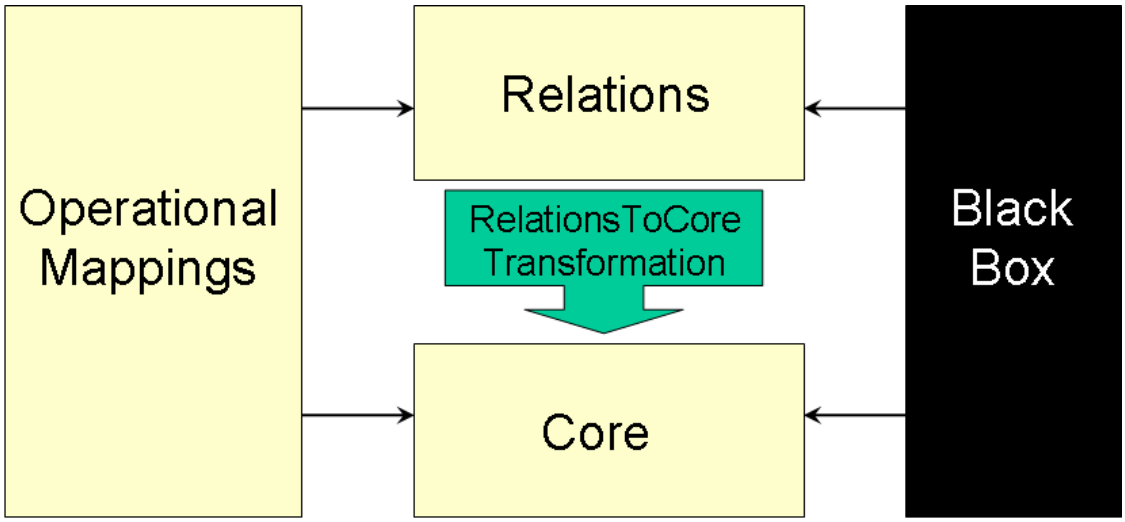
I.7.2.1 MOF QVT

The Object Management Group (OMG) has defined a standard for model transformations in Model Driven Architecture (MDA). This standard is called Query, Views, and Transformation (QVT). Queries are performed on input models to find specific elements or transformation. An example of a query could be to find all classes in a model. Views are models derived from other models. The result of the query is a kind of view. Transformations are, as explained above, the process of transforming an input model to an output model. A more detailed description of QVT can be found at http://umt-qvt.sourceforge.net/

Queries are performed on input models for finding specific elements or information. Example: Return all classes. Views are models derived from other models. The result of a query

is a kind of view. Transformations takes a model as input and creates a new model Example: PIM->PSM. Mappings are defined in Relations language or Core language. Uses Queries and Views.

To define a transformation the abstract syntax must be defined as a metamodel in MOF. Concrete syntax expressed as text (program code) or models. Transformations are defined in two variations: Declarative or imperative.



- Relations
  - Declarative specification of the mapping between MOF models
  - Equivalent with Java code (high-level)
- Relations to Core transformation
  - Equivalent with compiling Java code into byte code
- Core
  - Equivalent with Java byte code (low-level)
- Operational mappings
  - Standard language for defining Relations (or Core) in an imperative way
- Black box
  - Offers the possibility to plug-in code from any programming language with a MOF binding

Given a defined metamodel for source and target, metamodels must be well-defined according to MOF (models on level M2). One can define transformations between these two worlds in a general, standardized manner. The transformation can be used on all instances of the source metamodel. The result will be an instance of the target metamodel.

1.7.3 Transformation tehnologies

There are multiple technologies for mapping and transformation:

- Java
- IBM Model Transformation Framework (MTF)
- Atlas Transformation Language (ATL)
- Not many QVT-based ones: QVT support in Borland Together Architect 2006

I.7.3.1 Java

Technologies such as MDR and EMF allow for generation of Java API toward arbitrary metamodels. Using these APIs mappings can be defined in a Java program. The transformation is performed by running the Java program on a model instance given as input. The drawback is that API generation for metamodels is needed. The pros are a well-known language for mapping definition.

I.7.3.2 IBM MTF

MTF was developed in order to experiment with QVT related concepts,. MTF is not QVT compliant. MTF rules describe the relationships between the input and the output metamodel. Provides functionality to define mappings between metamodels and execute the model transformations.

I.7.3.3 Atlas Transformation Language (ATL)

The ATL project aims at providing a set of transformation tools for GMT [GMT: http://www.eclipse.org/gmt/]. These include a transformations repository, some sample ATL transformations, an ATL transformation editor plug-in, an ATL transformation engine, etc.

The Atlas Transformation Language (ATL) has been developed by the ATLAS team, INRIA. It is a hybrid language (a mix of declarative and imperative constructions) designed to express model transformations as required by any MDA™ [MDA] approach (see the QVT RFP [QVTRFP]). It is described by an abstract syntax (a MOF meta-model), a textual concrete syntax and an additional graphical notation allowing modelers to represent partial views of transformation models. A transformation model in ATL is expressed as a set of transformation rules. The recommended style of programming is declarative. Transformations from Platform Independent Models (PIMs) to Platform Specific Models (PSMs) can be written in ATL to implement the MDA™ approach as suggested by the OMG.

Eclipse has been used as an IDE for ATL, with advanced code edition features (syntax highlighting, auto-completion, etc.). ATL will provide a context in which transformation-based MDA™ tools can be designed and implemented for Eclipse.
The ATL project provides a complete environment for developing, testing and using model transformation programs through the following items:

- A transformation repository supporting the creation of a library of transformations ranging from simple examples to fully reusable components. A prototype repository (RAS-like [RAS]) already exists that stores transformation components in compressed archives (ZIP files) including a meta-data description in the form of an XML file. Components can be: transformations (written in ATL, written in another language or composite transformations using others), meta-models (a transformation depends on the transformation meta-model, the input and output meta-models) and models (input/output samples).

- A source code editor for transformations adapted from the Eclipse source code editor. Several levels of implementation are possible, from a simple text editor to a full-featured one (including syntax highlighting, auto-completion, etc.). Users will be able to launch transformations from the IDE, which could interpret error messages and use them to point out the problems to the modeler.

- A debugger will also be integrated, in order to complete the Eclipse IDE for ATL.

- And finally a transformation engine (for ATL v0.2) has been released as part of the GMT project.

Although the QVT RFP asks for a transformation language for MOF models, the ATLAS team thinks that making other metametamodels usable, provided they are semantically close enough to MOF, would add considerable value to a transformation language. This metametamodel independence would even be very interesting among OMG standards, MOF 2.0 being different from MOF 1.4.

Finally, the ATL framework also integrates the notion of "technical spaces" [Kurtev0001]. Although mainly intended to deal with MDA™ models (based on MOF meta-models and accessible via XMI [wwwXMI] or JMI [JMI10][JMIfaq]), this framework should also handle other kinds of models from different technological spaces (e.g. Java programs, XML documents, DBMS artifacts, etc.). To this end, what is needed is a collection of adaptable "injectors" and "extractors" to complement the library of MDA™ transformation components. Models, meta-models, transformations, injectors and extractors are examples of MDA™ components that will be handled as uniformly as possible by the ATL repository.

-
    - Hybrid
        - Declarative => transformations based on simple mappings
        - Imperative => for complex mappings
    - Based on OCL
- Documentation
    - http://www.eclipse.org/gmt/atl/doc/

An ATL transformation program is composed of rules that define how source model elements are matched and navigated to create and initialize the elements of the target models. The ATL programs for model to model transformation are called modules. A module is composed of :

- Header
    - transformation name
    - variables of source and target models
- Import
    - libraries to be imported
- Helpers
    - define (global) variables and functions
- Rules
    - describe the transformation from a source model to a target model

I.7.4 Example and Tutorials , ATL - Book2Publish, PIM4SOA to XSD

Useful examples and tutorials on use of ATL can be found here,

http://www.eclipse.org/m2m/atl/

and at the INF5120 course Wiki, http://set.sintef.no/inf5120/index.php/Tutorials

I.7.5 Model Transformation (MT)

Within the context of MDD, model transformation is the primary operation on models that is talked about, however it is not the only one. Operations such as model comparison, model merging etc are also considered, although these could be seen as particular types of model transformation. This section focuses further on the SoA of Model transformation.

The concept of model transformations existed before QVT and even before MDA, the following topics each address some aspect involving the notion of transforming data from one form to another.

- Compiling Techniques [Aho etal]
- Graph Grammar/Transformations [EEKR99]
- Triple Graph Grammars [Schurr]
- Incremental Parsers [Ghezzi, Mandrioli]
- Viewpoint framework tools [Finkelstein etal]
- Databases, Update queries
- Refinement
- XML, XSLT, XQuery [w3c]

Even simple straight forward programming is generally done as a means to transform data. This becomes more stylised when programming patterns such as the Visitor pattern [gof] are used as a way to visit data in one model and create data in another.
Some techniques, such as those presented in [Akehurst etal] [qvt-partners] [dstc-qvt], base the transformation language on the notion of relations. However, this too is a new application of the old ideas as originally applied (for example) in the fields of databases (e.g. Update Queries) and System Specification (or refinement) using the formal language Z (which is heavily dependent on the notion of relations and their use for the manipulation of data).
The interesting aspect of the MDD approach to transformation is the focus on:

- Executable specifications; unlike the Z approach.
- Transforming models; models being viewed as higher level concepts than database models and certainly higher level than XML trees.
- Deterministic output; the main problem with GGs is that they suffer from non-deterministic output, applying the rules in a different order may result in a different output.

A general depiction of a transformation and the necessary relevant specifications is shown in the figure below.

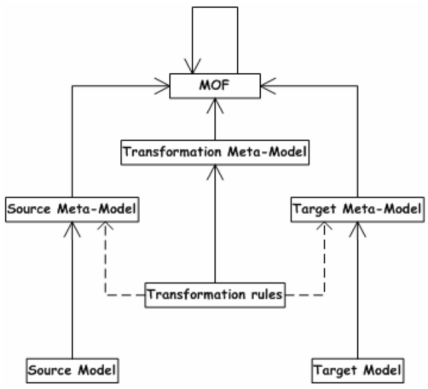


Figure 2 Transformation with specifications

Transformation of one model into another model is the central concept of model-driven development. Specifying and implementing model transformations is thus a field of active research.

Model-transformation can be tackled using general-purpose programming languages; however, specialized transformation languages can significantly ease the implementation of transformations. In the following we classify the various approaches by the programming paradigm that is used:

- functional languages,
- pattern-based languages,
- object-oriented languages,
- hybrid languages, and
- graphical languages.

I.7.5.1 Functional Languages

Functional programming languages are one way to implement transformations [HJGP99]. Here transformations are carried out by applying transformation operators on models. With the normal filter, map, and reduce functions these operators can be concatenated to more complex operations [WMH00]. One advantage of using functional languages is that the developer does not have to deal with model traversal. Furthermore, existing operators can be reused. However, the approach inherits some problems from functional languages. Pure functional programs are usually free of side-effects. An efficient implementation of model transformation thus requires the use advanced concepts such as monads [Jon01] or uniqueness-types.

I.7.5.2 Pattern-based Languages

As the standard serialization format for models (XMI) is based on XML, some researchers [DHO01, PBG01, PZB00, VVP02, KH02] suggest transforming the XML document instead of the graph representation of the model. XSLT [W3C99b] can be used as a general purpose transformation language for XML documents. XSLT uses the concept of patterns. It traverses the XML tree structure and tries to match its pattern at every node of the tree. If the pattern matches, a certain transformation rule is applied. While this concept works very well for trees, it is difficult to adapt it to arbitrarily shaped graphs. The problem of sub-graph matching in arbitrarily shaped graphs is known to be NP-complete [GJ90].
Using XSLT has severe drawbacks. XML documents are trees structures only. Most models, especially behavioral models such as state-machines or activity diagrams, are rather complex graph structures. It is possible to cross link elements outside the tree structure using standards such as XPointer [W3C02] or XLink [W3C01]. However, XMI does not use them. With XPath [W3C99a] model edges that are not part of the serialized tree can be traversed. However, this leads to very complex implementations. Hence, XSLT and similar approaches are considered to be too difficult to use without a front-end language [PBG01].

I.7.5.3 Object-oriented languages

Object-oriented languages are currently the first choice programming paradigm when looking for a general purpose programming language. Using them for model transformation is straight forward, if an object-oriented API to access the model is available. The drawback of OO languages is that the programmer has to write code for traversing the data structures [WUG2003]. Traversing trees or even arbitrary graphs is non-trivial. It often results in deeply nested loops and increases the possibility of bugs such as infinite loops or infinite recursions. Other approaches like functional languages or pattern-based languages hide the model traversal from the developer.

I.7.5.4 Hybrid Languages

Combining the advantages of different programming paradigms can be beneficial for model transformations. The Object Constraint Language [OMG03i, OMG03d] has been developed to describe constraints on instances of models. OCL is a hybrid language merging functional and OO concepts. OCLs syntax and handling of data-types are close to that of object-oriented languages. But it also features functional-style constructs such as forall and select that are conceptually close to the functiona concepts of map, filter, and reduce. As OCL has been designed to work on models, it

seems most suitable for the problem of model transformation. However, OCL is a constraint language and designed to be free of side effects.

The UMLAUT transformation framework [Tri01] is a research project that builds model transformations on top of the upcoming OCL2 [OMG03d] and the action semantics standard [SPH+01, OMG03d] that allows OCL2 scripts to add and remove objects and to change their properties. The VisualOCL project [BKPPT01, KTW02] equips OCL with a graphical notation. This may ease the creation and understanding of OCL because its textual notation is very compact. VisualOCL's notation does, however, not provide new concepts. In [BKPPT02] VisualOCL is combined with graph transformation theory. This way OCL is extended with operational semantics.

KASE [WUG03] uses Python as its scripting language. Python scripts can be used to implement model transformation. Python [Pyt03] is an object oriented language that has been extended with elements of functional programming. Python is used in commercial tools, too. For example, the MDA tools of Interactive Objects [Int03] build on Python when it comes to scripting.

I.7.5.5 Graphical languages

Textual languages are usually not the ideal media for graphs, which inherently have a non-textual, hence graphical, notation. AGG [TFS03, EEKR99, Tae99] is a tool for editing and transforming graphs. Applying AGG to model transformations is in theory possible although AGG has not been designed with UML in mind. However, AGG supports for higher-order graphs. It could handle association classes in the meta model. Higher-order graphs allow for edges between edges. This way, nodes and edges can be handled in a uniform way [LB93].

The UML itself can be extended with graph transformation capabilities. [FNTZ98] uses story diagrams that show how links between objects are created or removed, how objects are created or destroyed or how the attribute values of the objects change. The approach taken by [HE00] is a merge of UML object diagrams and graph transformations. In so far it is comparable to Story Diagrams. Similar to [FNTZ98] they use ideas from graph transformation theory to model the behaviour of a system. The approach acts on the object level (M0), too, just like Story Diagrams. However, the MDA is not concerned with the object level. Transformations address the models on M1 level themselves. A model can be treated as an instance of the meta model (M0 objects are instances of the M1 classes). Story Diagrams, the approach of [HE00], or AGG could be used for model transformation, too. This means that that transformation developer would have to specify the transformation in terms of meta-model objects. However, a shift of the meta models would ideally be accompanied with a shift of the notation. An M1 instance of an M2 meta class has a special notation in the UML that depends on the meta class. A shift of notation cannot be easily accomplished since the UML notation extension introduced for Story Diagrams works for UML object diagrams only.

All the previous techniques modify graphs. A start graph is transformed several times to reach a final derivation. Usually, this final derivation still shares a common subset with the start graph, which is the case for all examples found in [AEH+99, FNTZ98, HE00]. In the MDA it is less common to morph a PIM step by step until it is a valid PSM. Even worse, this is usually not possible at all since the PIM is based on another meta-model.

VIATRA [VVP02, CHM+02, VGP01] is model transformation approach based on graph transformation. VIATRA has been designed to transform a model from one modelling language to another one. The overall goal of VIATRA is to check the correctness and reliability of a design. The design for example an UML state-machine is transformed into a more precise model better suited for formal reasoning, for example Petri Nets. Existing tools for Petri Nets can be applied. The result of these tools is back-annotated to the UML state machine model. To achieve this back-annotation,

VIATRA builds a reference graph between the input and the output model during the transformation process.

Kafka [WUG03] is a graphical model-transformation language that is based on graph transformation. It supports transformation of models from one meta-model (PIM) to models of a different meta-model (PSM). During the transformation Kafka builds a reference graph between the both models. With this reference graph roundtrip engineering is possible. Changes that the developer made to the PSM are not lost during a subsequent model-transformation step. Kafka reuses the notation of the source and target meta-model. A diagram for the transformation of a component (PIM) into a class (PSM) would show a component and a class in their normal notation. A detailed knowledge of the meta-models is thus not required for specifying transformations.

I.7.6 QVT and Model Transformation Tools

Much work on model transformation is being driven by the OMG's call for proposals on Queries, Views and Transformations (commonly known as QVT) [qvt]. There are a number of submissions to the standard with varying approaches, a good review of which is given by [GGKH03] along with some other (independent) approaches such as YATL [Patrascoiu], MOLA [Kalnins etal] etc. and earlier work such as [Akehurst].
There are a set of requirements, given in [GGKH03], of which the multiple approaches each address a different subset. As yet there is no approach that addresses all of the requirements.

Some MDA oriented tools and platforms are introduced in this section. Some of these tools and platforms are open source software such as EMF and UMT. The others are commercial products such as ArcStyler, Rhapsody and OptimalJ. More MDA tools and platforms can be found in modelbased.net [Modelbased].

I.7.7 MOFScript for Model to text

The MOFScript Eclipse Plug-in is a text generation tool, developed to provide a user-friendly code generation tool for arbitrary metamodels. The language has been submitted to the OMG process for Model to Text Transformation[1]. The MOFScript tool is now part of the Eclipse GMT (Generative Modeling Technologies) Suite of tools, a collection of Eclipse-based tools supporting model-driven engineering[2].
The MOFScript tool provides the means of editing, compiling, and executing model-to-text transformations written in the MOFScript language. The tool is packaged as an Eclipse plug-in, which provides editing capabilities with syntax highlighting and content assistance for the language. The tool can be installed through the update site www.eclipse.org/gmt/mofscript/update/.

I.7.7.1 MOFScript Architecture

The MOFScript tool is an implementation of the MOFScript language and it is developed as two main logical achitectural parts: *Tool components* and *Service components* (see Figure 3). The *Tool components* are end user tools that provide the editing capabilities and interaction with the services. The services provide capabilities for parsing, checking, and executing the transformation language. The language is represented by a model (the MOFScript model), the model is an Eclipse Modeling Framework (EMF)[3] model that is populated by the parser. This model is the basis for semantic checking and execution.

---

[1] MOF Model to Text Transformation Language RFP, OMG document ad/04-04-07, http://www.omg.org/cgi-bin/apps/doc?ad/04-04-07.pdf
[2] Eclipse Generative Modeling Technologies (GMT) http://www.eclipse.org/gmt/
[3] Eclipse Modeling Framework • By Frank Budinsky, Dave Steinberg, Ed Merks, Ray Ellersick, Timothy J. Grose Published Aug 11, 2003 by Addison Wesley Professional. Part of the The Eclipse Series series. ISBN-10: 0-13-142542-0 ISBN-13: 978-0-13-142542-2

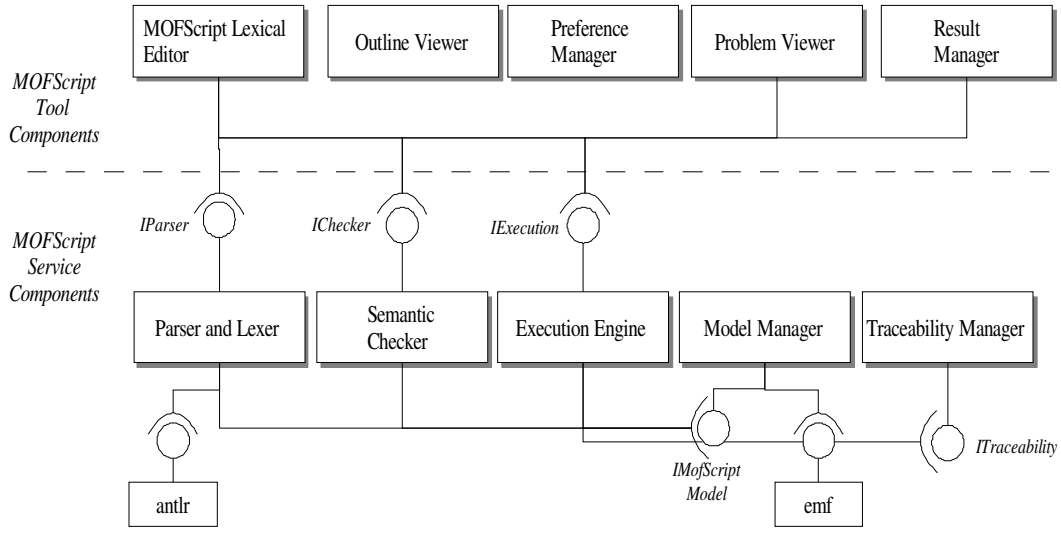


Figure 3 MOFScript Architecture

The Service Components consist of these component parts: The *Model Manager* is an EMF-based component which handles management of MOFScript models. The *Parser and Lexer* are responsible for parsing textual definitions of MOFScript transformations, and populating a MOFScript model using the Model Manager. The parser is based on antlr [4]. The *Semantic Checker* provides functionality for checking a transformation's correctness with respect to validity of the rules called, references to metamodel elements, etc. The *Execution Engine* handles the execution of a transformation. It interprets a model and produces an output text, typically to a set of output files. The *Traceability Manager* handles the traceability between generated text and the original model, aiming to manage changes to a model or its target text.

I.7.7.2 The user interface

The MOFScript user interface is provided through Eclipse's editor functionality. It encompasses, as depicted in Figure 4, a lexical editor, an outline viewer, a console and problems view. In addition, a configuration manager and a new file wizard can be invoked.

[4] ANTLR Parser Generator: http://antlr.org/

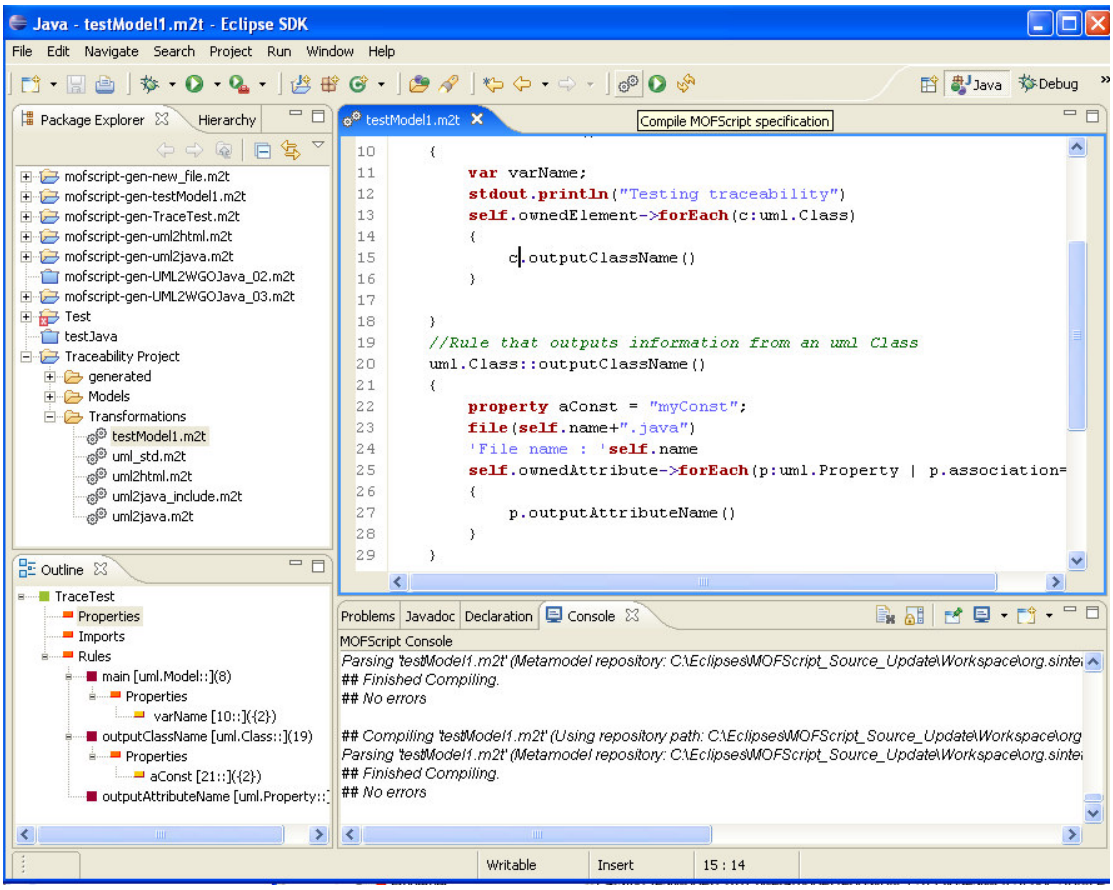


Figure 4 MOFScript user interface

The lexical editor provides syntax highlighting for special keywords and comments, this functionality makes the transformation easy to read; green for comments, keywords in red and text and escaped output in blue. The user interface also provides buttons for compiling, executing and re-executing the previous transformation. MOFScript also has extensive code completion support, both for metamodel references (navigation) and for user-defined rules found in the transformation module or an imported library. This makes writing transformations in MOFScript simple and effective[5],[6].

Further information on MOFScript can be found at the course Wiki,

http://set.sintef.no/inf5120/index.php/Tutorials

I.7.8 ArcStyler

ArcStyler is a commercial MDA tool from Interactive Objects [Modelbased]. ArcStyler is a cross-platform, standards-compliant, environment, fully implemented in Java, for the rapid design, modelling, generation, deployment and management of high-quality, industrial strength applications of any size for architectures based on Java/J2EE and .NET as well as custom infrastructures and existing legacy platforms [IO-Software].

[5] MOFScript White Paper: J. Oldevik, G.K. Olsen
http://www.modelware-ist.org/index.php?option=com_remository&Itemid=79&func=fileinfo&id=118

[6] MOFScript User Guide: http://www.eclipse.org/gmt/mofscript/doc/MOFScript-User-Guide.pdf Oldevik, Jon

ArcStyler provides a comprehensive, architecture-driven solution for end-to-end model-driven development of modern, component-based applications. By assisting developers with important architectural tasks, the ArcStyler simplifies and expedites the entire development life cycle, from the platform-independent business model to platform-specific refinement and optimized code generation for the leading application servers. This approach is currently being standardized as Model Driven Architecture by the OMG [IO-ArcStyler].

ArcStyler is described by Interactive Objects as an 'Architectural IDE', which attempts to encapsulate similarities between software architecture and industrial architecture. The most significant characteristic in the ArcStyler is its positioning above traditional, programming IDEs (in fact such a tool is a component of ArcStyler) [Charlesworth].

ArcStyler is made up of a number of tools and modules. Among them, MDA-Cartridges are pluggable MDA automation engines and contain the entire model-to-model and model-to-code transformation logic as well as model verifiers [IO-Cartridges]. Through the use of MDA Cartridges, ArcStyler does not require developers to have in depth platform knowledge. The most important aspect is the use of MDA Cartridges to ensure that the generated code is optimised for the environment and also conforms to the coding styles and more specific requirements of the organisation. This allows solutions to be configured for all of the major application servers (J2EE, .NET, CORBA) as well as for custom environments (COBOL, z/OS, RPG) or a mixture of these. Interactive Objects develops and supports a number of MDA Cartridges providing support for leading application servers and their descendants, including [Charlesworth]:

- BEA WebLogic Server.
- IBM WebSphere (NT and z/OS).
- IONA E2A Platform.
- Borland Enterprise Server.
- The JBoss Application Server.

Three editions of ArcStyler are now available [whitepaper]:

1. The ArcStyler Web Edition, which provides MDA automation and MDA Cartridges for Web technologies including Web services, XML, .NET and J2EE.
2. The ArcStyler Enterprise Edition, which enhances the Web Edition with MDA support for all levels of an n-tier architecture. It also adds the Business Object Modeller, the Refinement Assistant, as well as an explorer for the common model repository.
3. The ArcStyler Architect Edition comprises the Enterprise Edition plus the MDA Development IDE. This is used by the organisation to develop MDA support for its own architectural style through visual development or extension of MDA Cartridges according to a well-defined Cartridge Architecture. This is a key capability and allows for technical knowledge to be captured, incorporated into an MDA Cartridge, and communicated throughout the business.

ArcStyler supports the creation of, and relationships between, numerous UML models.

I.7.9 Rhapsody

Rhapsody, by I-Logic, is the industry's leading UML 2.0 based Model-Driven Development environment for systems and software engineering [Rhapsody].
The philosophy of Rhapsody has always been the generation of platform-independent models that map onto many different computing platforms, long before the OMG's inception of the MDA initiative. Rhapsody consists of several collaborating parts [Douglass]:

- Model-Entry System – the developer enters in the PIM using standard UML diagrams
- Model Compiler – the developer generates the source for the selected language (C, C++ or Java) and compiler
- Model Tester –allows the tester to stimulate and monitor the execution of the PIM-generated application on the host or target platform

- Framework – a real-time PIM framework, provided by Rhapsody, that runs underneath your PIM
- OS-Dependent Adapter – a lightweight OS-specific adapter layer that handles interaction with the underlying RTOS

Rhapsody is designed to support complete model-based iterative life-cycle. The following key enabling technologies are used in Rhapsody for effective model-based development including the common automation and traceability deficiencies: model-code associativity, automated implementation generation, implementation framework, model execution and back animation, and model-based testing [Gery, Harel, Palachi].

I.7.9.1 Model-code Associativity

Model-code associativity is a key enabler for software developers to effectively leverage the benefits of model-based approaches, but without compromising the benefits of direct access to the implementation language/platform [Gery, Harel, Palachi]. In Rhapsody, the implementation language is augmented by an execution framework, the interface between the implementation language and the modelling language abstractions. Detailed behaviours are not written in modelling language but in the target implementation language. Code translated from the model by using common translation patterns for UML's abstract construction and by attaching notes and constraints to the resulting code. Rhapsody supports to trace the code resulting from certain model elements and vice versa. The changes in the model are instantaneously reflected in the code view and vice versa.

I.7.9.2 Automated implementation generation

Automated implementation generation is the core support of Rhapsody. Implementations can be generated not only from structural semantics but also from all the behavioural semantics specified in the UML model. These include system construction, object life-cycle management (construction and deconstruction), object behaviour as specified by statecharts or activity-graphs, as well as methods directly specified by the implementation language [Gery, Harel, Palachi]. Codes are generated from model artefacts based on generation rules and predefined parameters for each model metaclass. Rhapsody implementation generator supports implementation languages such as C++, C and Java, and component frameworks COM and CORBA.

I.7.9.3 The execution framework

The execution framework is an infrastructure that augments the implementation language to support the modeling language semantics [Gery, Harel, Palachi]. APIs based on the implementation language are provided in the execution framework to support for manipulation of model at model abstraction level. A set of architectural and mechanistic patterns are used in this framework to support modelling abstractions. The execution framework API serves as an abstraction layer used by the code-generator to facilitate model semantics in the context of a particular implementation language [Gery, Harel, Palachi].

I.7.9.4 Model execution

Model execution is a key enabler for effective model-based development. Rhapsody uses the model-animation technique, a runtime traceability link between the implementation execution and the runtime model [Gery, Harel, Palachi]. This technique eases the mapping UML design model and code implementation. It also enables shorter iterations of model-implement-debug cycles.

I.7.9.5 Model-based Testing

Model-based testing provides the ability to specify and run tests based on the specifications in the model, as well as the ability to pinpoint defects from the test results by visualizing points of failure within the model [Gery, Harel, Palachi]. Rhapsody specifies tests using scenarios specified in the

model. The tests are run by driving and monitoring the model execution. Rhapsody provides functions to detect defect, to fix defect and verify by rerunning the test.

I.7.10 OptimalJ

OptimalJ from Compuware is an advanced development environment that makes complete use of the Object Management Group (OMG) Model Driven Architecture (MDA) to enable rapid design, development, modification and deployment of J2EE business applications [OptimalJ MDA]. OptimalJ delivers high productivity and consistency throughout the development cycle, while helping to reduce technical complexity for development teams [OptimalJ Product Preview].
A Java application is constructed in OptimalJ through a series of multi-level models and patterns. At model level, the OptimalJ include the Domain Model (PIM in MDA), the Application Model (PSM in MDA) and the Code Model (Implementation Model in MDA).

The Domain Model defines the business domains without application details. There are two parts in domain models: a class model describing the static structure of the applications data, and a service model describing some behavioural aspects. The domain model is based on the Meta Object Facility (MOF) and Common Warehouse Model (CWM) in UML.

The Application Model presents a fairly high level abstraction of J2EE concepts, and is created from the domain model at the user's behest by OptimalJ. There are three application models: a database model, an EJB model and a web model [Evaluation OptimalJ]. The Code Model defines the generated application code transformed automatically from the Applicatoin Model [OptimalJ Product Preview]. OptimalJ can generate all the relevant Java, JSP, XML files etc., necessary for a J2EE application [Evaluation OptimalJ].

Two types of patterns are distinguished in OptimalJ [OptimalJ Product Preview]:

- Transformation patterns – They are used in transforming OptimalJ's Domain Model into the Application Model, and Application Model into the Code Model.
- Functional patterns – They are code templates which are predefined functionality. Development can be sped up and errors can be reduced through quality-proven models or code in application.

There are three editions of OptimalJ are available [Evaluation OptimalJ]: OptimalJ Developer Edition, OptimalJ Professional Edition and OptimalJ Architecture Edition.

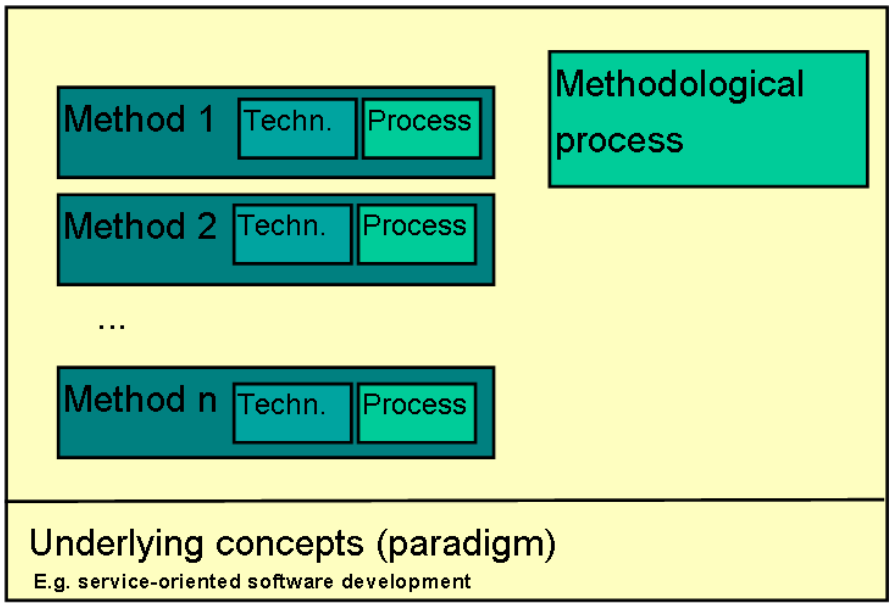
I.8 Method Engineering

I.8.1 Methodology definition

A method is a systematic process, technique or mode of inquiry that is used to aid in the creation of a satisfactory software product. [Blum94]. In a model-driven development approach we use a method to produce models. Method include technique and process. For example a CRC method includes CRC technique and CRC process.

- Technique: A specific construct supporting a method
- Process: A sequence of actions leading to some result

A methodology is a body of methods. It is meant to support all software development phases.



As we can see in the figure a methodology consists of a set of methods, each having their own technique and a process. A methodological process guides the usage of these methods. Methodologies typically build upon underlying concepts (or paradigms). Examples of such paradigms are: Object-oriented software development, component-based software development and service-oriented software development.

I.8.1.1 Principles for modern software development

Software development methodologies has evolved gradually over the past years. A modern software development methodology typically follows a set of these principles:

- Architecture-centric, e.g. service-oriented architecture
- Iterative and incremental process
- Service- and component-orientation
- Manage a highly dynamic environment.
    - Processes to support iterative and incremental development.
    - Software system mush be designed to be easy to change.
- Model-based
- "Round-trip" engineering
- Divide and conquer
- Quality check
- Configurable process

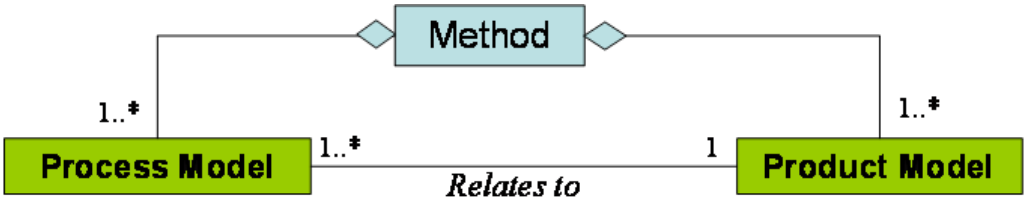
I.8.1.2 Process

The process depends on the type of system. It is very rare that a brand new system is being developed. Most of the cases developers are involved in re-engineering or modification. Re-engineering is a radical process of re-designing an old system while modification is a process of fixing a major problem or changing a feature. Another typical process scenario is to add new functionality to an existing system, by developing and integrating a new module (e.g. component or service).

1.8.2 Existing methodologies

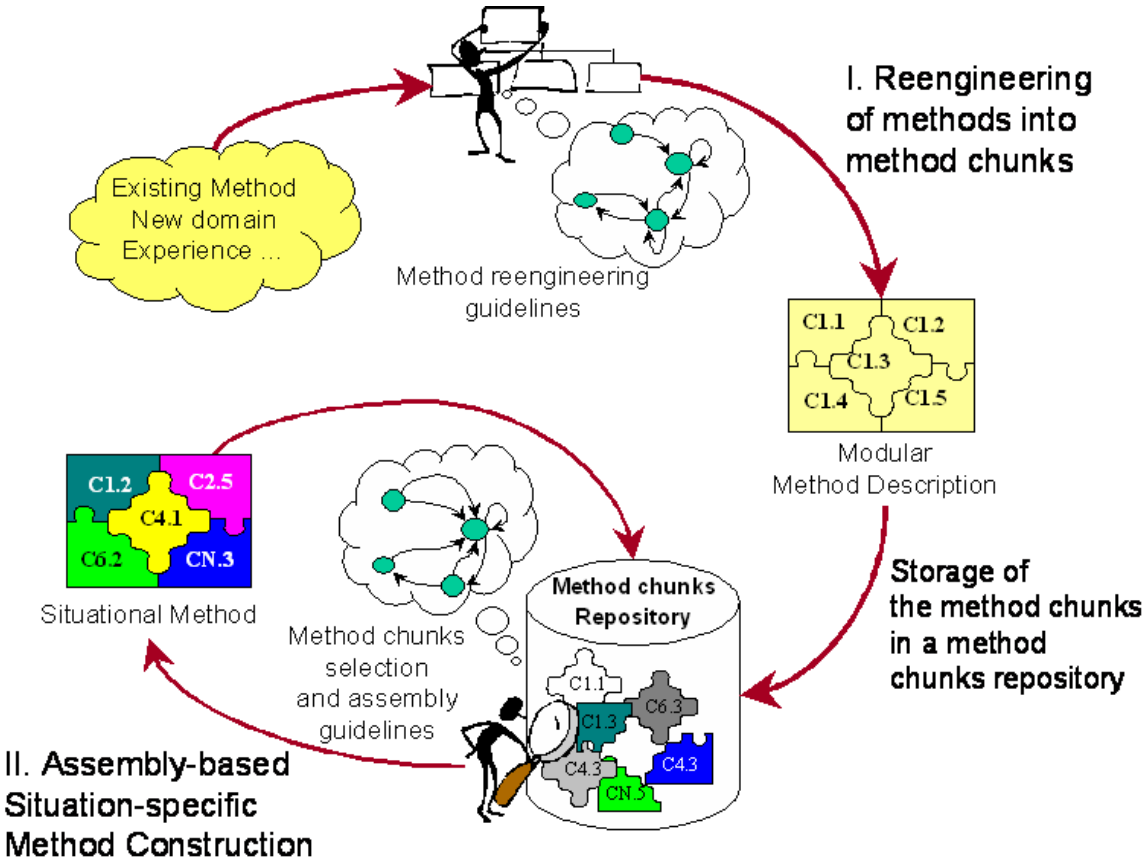
- The Rational Unified Proces(RUP) is an iterative software development process created by the Rational Software Corporation. It is an adaptable process framework that describes how to develop software effectively using proven techniques.
- KobrA methodology for modeling architectures is a systematic approach to using UML that is refreshingly simple and straightforward. One of the problems on components is that the term "component" is so heavily overloaded . To overcome this, KobrA has introduced a new term "Komponent". Komponents are the "logical" components that represent the logical building blocks of a software system.
- Catalysis is another UML based method for object and component based development.
- Select Perspective is a methodology in agile modelling that uses multiple, best-of-breed modelling techniques, both text and diagram-based.
- The Object Oriented Role Analysise Method (OOram) is a method, based on the concept of role, for performing object-oriented modeling. OOram is a precursor for the Unified Modeling Language (UML). (Developed by profesor Trygve Reenskaug).
- Some other lightweight methodologies are eXtreme programing (XP), Adaptive Software Development (ASD), SCRUM – an agile method for project management, and Crystal Clear – an agile developing method.

1.8.3 Method engineering



From the engineering perspective, a method is made up of a set of product models and a set of corresponding process models. A product model represents the concepts that are used in the method, relationships between these concepts as well as constraints that they have to satisfy. A process model represents the way to accomplish the development of the corresponding product.

I.8.3.1 Method engineering process



In a method engineering process there are two foci:

1. Re-enginering of existing methodolologies into smaller methods or method chunks and describe the methodology as a set of configured modules. These method chunks will be stored in a method chunks repository.
2. Assembly of situation-specific methods based on existing method chunks.

I.8.3.2 Method chunk

A method chunk is an autonomous and coherent part of a method supporting the realisation of some specific system development or management activity. Such a modular view of methods favours their adaptation, configuration and extension. Moreover, this view permits to reuse chunks of a given method in the construction of new ones.

I.8.3.3 Problems and opportunities

There exist no authoritative compilations of method chunks that can be assembled to fit particular project contexts, and such that deliverables of applying one method chunk, can be mapped to inputs of another method chunk. Method chunks are rarely presented as elements that are separated from the problems solved with them, or from the cases used to illustrate their application. There is no agreed taxonomy of method chunks. Methods are in the heads of people, and not yet in the services of systems. There are no services to assess which methods to use in a given situation, given the bounded rationality of the engineer, often, sub-optimal methods are selected, making projects expensive and crippling system development.

However, there exists a near-consensus in the Method Engineering community, on the requirements for a method-chunk repository. There exists a wealth of architectural styles (service oriented, agents,...) that can assist the development of an MCR. Platforms such as

ATHENA's MPCE are offering infrastructural services, needs-awareness and proto-communities that are conducive for the MCR development and test.

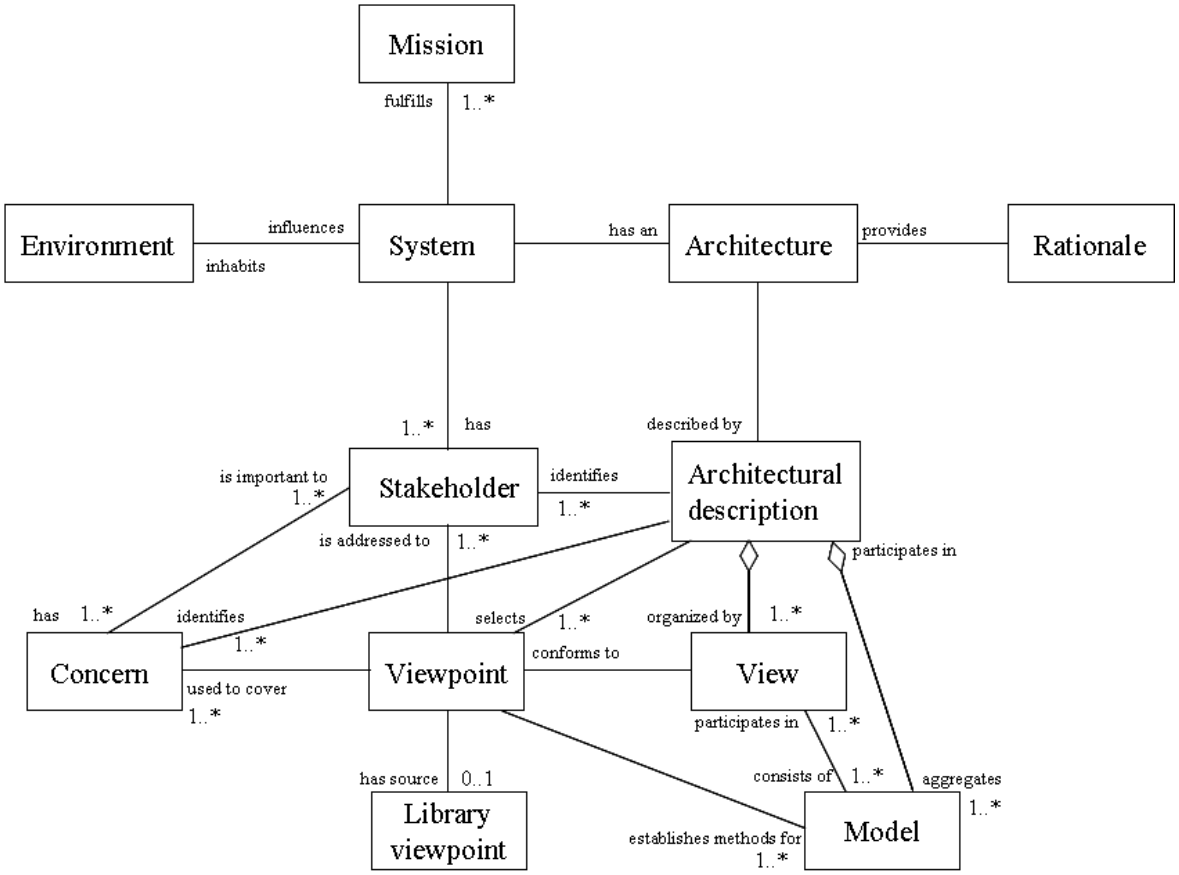
To support method engineering we need a modelling and execution platform. This platform must provide use case users, developers, method chunk managers and method chunk developers appropriate workplaces with services. We also see the need for both a use case repository and a method chunk repository.

1.8.4 Software architecture

Even if it is used by many, the term "architecture" has no well established definition. Nevertheless, in the field of software engineering there is no shortage of more or less overlapping definitions (see for instance www.sei.cmu.edu/architecture/definitions.html). Here we present one consistent set of definitions targeting architectural descriptions for software-intensive systems, namely the IEEE Std 1471-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. This recommended practice seeks to become a common frame of reference within which to codify common elements between different architectural description initiatives, and has become influential and used as a baseline for architectural description frameworks, for instance within OMG. It reflects generally accepted trends in practices for architectural description and provides a frame of reference within which future developments in software architectural technology can be deployed.

According to the recommended practice, software-intensive systems are those complex systems "where software contributes essential influences to the design, construction, deployment and evolution of the system as a whole". The purpose of IEEE Std 1471-2000 is to facilitate the expression and communication of architectures and thereby lay a foundation for quality and cost gains through standardisation of elements and practices for architectural description of software-intensive systems. This is in contrast to past attempts at architecture description where only hardware-related architectural aspects were addressed. With increasingly complex software, architectural integrity of the software should also be addressed and the recommended practice facilitates this.

The figure shows the conceptual model of architectural description as defined in IEEE Std 1471-2000.



Starting with system, it is defined to be “a collection of components organized to accomplish a specific function or set of functions.” For the purposes of the recommended practice, “the term system encompasses individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, whole enterprises, and other aggregations of interest.” From this it follows that anything can be a system as long as it fulfils some purpose (i.e., accomplishes function(s)) and one chooses to view it as a whole.

A system inhabits an environment, while the environment of a system can influence that system. The environment, sometimes referred to as the context, “determines the settings and circumstances of developmental, operational, political, and other influences upon that system. The environment can include other systems that interact with the system of interest, either directly via interfaces or indirectly in other ways. The environment determines the boundaries that define the scope of the system of interest relative to other systems”. Essentially, one draws a line between the system of interest and anything outside that system that influences it in some way. This line is the interface between the system and its environment.

A system has one or more stakeholders. A stakeholder has one or more concerns relative to the system. Concerns are “those interests which pertains to the system’s development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.” Typical concerns a stakeholder can have relative to a system are functionality, performance, security, reliability, safety, etc.

A system exists to fulfil one or more missions in its environment. The existence of a system has a purpose; it should meet one or more objectives of one or more stakeholders. Often some of these objectives coincide with enterprise objectives so that using the system is an efficient use of resources in the enterprise. So far, the terminology presented has only been related to systems and their environments. However, most of IEEE Std 1471-2000 is concerned with architectural descriptions, and in the following terminology related to this are presented.

A system has an architecture and this can be described in an architectural description. Note the distinction between the architecture of a system, which is conceptual, from the description of this architecture, which is concrete. Architectural description is defined as “a collection of products to document an architecture”. The architectural description can be divided into one or several views. Each view covers one or more stakeholder concerns. View is defined as “a representation of a whole system from the perspective of a related set of concerns”. A view is created according to rules and conventions defined in a viewpoint. Viewpoint is defined as “a specification of the conventions for constructing and using a view. A pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis”. The distinction between view and viewpoint is analogous to that between a searchlight and what one sees using the searchlight as shown in the figure below.

I.8.5 MDA standard - SPEM

I.8.5.1 Software Process Engineering Metamodel (SPEM)

The Software Process Engineering Metamodel (SPEM) is a metamodel and a UML profile to describe software engineering processes.

- Identifies the typical concepts of a process (process, phase, role, model, etc.)
- Defines them using UML extensions (stereotypes applied to various elements: class, use cases, operations, etc.)
- Assigns a characteristic icon to each new item.

The SPEM representation of a process is based upon the collaboration between active entities called ProcessRoles that execute tasks called Activities on concrete entities called WorkProducts. SPEM defines notational icons or graphical representation for SPEM concepts that are represented by UML stereotypes.

I.8.5.1.1 Modelling constructs

The concepts defined in SPEM are:

- **WorkDefinition:** It’s a kind of operation that describes the work performed in the process. Its main subclass is Activity, but phase, iteration and lifecycle are also subclasses of WorkDefinition.
- **Activity:** It’s the main subclass of WorkDefinition. An activity describes any part of work executed or assisted by a ProcessRole like tasks, operations and actions. An activity may consist of atomic elements called steps.
- **Process:** A Process is a ProcessComponent intended to stand alone as a complete, end-to-end process. It is distinguished from normal process components by the fact that it is not intended to be composed with other components.
- **Document:** It is a specialization of a WorkProduct that contains information.
- **Guidance:** Indirect support to actual software developers or managers working with a process-centred software engineering environment through interpretation of an instantiated process model.
- **Phase:** A phase is a specialization of WorkDefinition such that its precondition defines the phase entry criteria and its goal (often called a “milestone”) defines the phase exit criteria. Phases can be sequential or can run in parallel.
- **ProcessPackage:** It is a special notation for packages in a SPEM context. A package is a cohesive collection of work products.

- **ProcessPerformer:** The process performer is the performer of higher-level aggregate WorkDefinitions that cannot be associated with individual ProcessRoles. ProcessPerformer represents abstractly the “whole process” or one of its components, and is used to own WorkDefinitions that do not have a more specific owner.
- **ProcessRole:** The process role is the performer of activities. Also defines responsibilities over specific WorkProducts, and defines the roles that perform and assist in specific activities.
- **UMLModel:** It is a specialization of a WorkProduct that allows to model static, dynamic and behavioural view of process.
- **WorkProduct:** A work product or artefact is anything produced, consumed, or modified by a process. It may be a piece of information, a document, a model, source code, a service provided to an end user and so on. A WorkProduct describes one class of work product produced in a process.

I.8.6 Eclipse technologies - EPF

I.8.6.1 Eclipse Process Framework (EPF)

The Eclipse Process Framework (EPF) presents a process management tool platform and conceptual framework for authoring, tailoring and deploying development processes.

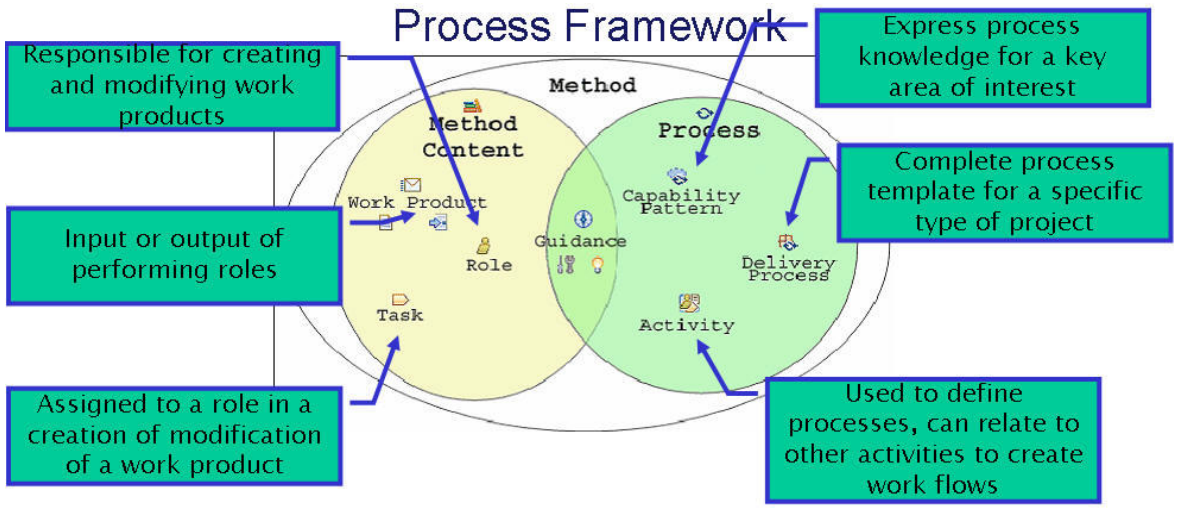
First release version 1.0 planned for September 2006

I.8.6.1.1 Goals

The Process Framework Project has two goals:

1. To provide an extensible framework and exemplary tools for software process engineering - method and process authoring, library management, configuring and publishing a process.
2. To provide exemplary and extensible process content for a range of software development and management processes supporting iterative, agile, and incremental development, and applicable to a broad set of development platforms and applications.

I.8.6.1.2 Process framework



Typically the word framework is used in the context of an environment one can use to define partial or complete solutions for a particular problem domain. In defining these solutions reusable structures are used as starting points or building blocks.

A process framework is a similar thing. It defines reusable method content, processes, building blocks and tools that can be utilized to define specific processes and methods. (EPF Composer ships with one or more of these process frameworks, to help the developer in the process, but as well is equiped with the ability to create a completely new method from scratch)th

Method content is primarily expressed using work producs, roles, tasks and guidance. Roles are identified in the development process and these represent a set of skills. Competencies and responsabilities in the development team. These roles are responsible for specific work products and for creating or modifying these work products, roles are assigned to perform tasks.

In addition to roles, tasks and work products EPF supports the addition of guidances in the form of whitepapers, concept description, guidelines, examples etc. Guidence is also present in the process representation.

In the representation of the process the main concept is the activity which can be nested to define breakdown structures and they can relate to each other to form work flows. Activities in EPF define to kind of processes: delivery processes and capability patterns.

Delivery processes represent a complete and integrated processs templete for performing one specific type of project.

Capability patterns are processes that express process knowledge for a key area of interest such as disipline or a best practice. They can be directly used by process practitioners to guide their work.

I.8.6.1.3 EPF Composer

EPF Composer is a tool platform for process engineers, project leads, project and program managers who are responsible for mainteining and implementing processes for development organizations or individual projects

Aims to:

- provide for development practitioners a knowledge base of intelectual capital that allows them to browse, manage and deploy content.
- provide process engineering capabilities by supporting processe engineers and project managers in selecting, tailoring, and rapidly assembling processes for their concrete development process.

EPF Composer allows you to take method content and to structure it in one specific way using a predefined schema. This schema is an evolution of the SPEM 1.1 OMG specification, which is currently being evaluated to become the 2.0 version of this specification. This method contents can overcome the limits of software engineering covering other design and disiplines such as bussiness transformations, sales cycles and so on. In EMF Composer they are represented as a construct of roles defining development skills and responsabilities for work products. Method contents can be found in books or publications on software engineering methods, best practices, standards, regulations or even homegrown methods.

In EMF Composer processes are typically expressed as workflows or breakdown structures. This framework provides oportunities that aims at supporting processes based on many different development approaches, development culture and development process representation.

Key concepts

- Redesigned tools for authoring, configuring, viewing, and publishing development processes.
- Just-in-time generation of publication previews
- Management of method content using simple form-based user interfaces.
- Intuitive rich text editors for content description
- Processes with:
    - breakdown structure editors
    - workflow diagrams
- Support for many alternative lifecycle models.
- Improved reuse and extensibility capabilities.
- Reusable dynamically-linked process patterns

I.9 Bibliography – Model Driven Development

[AEH+99] Andries, M. ; Engels, G. ; Habel, A. ; Hoffmann, B. ; Kreowski, H. J. ; Kuske, S. ; Plump, D. ; Schuerr, A. ; Taentzer, G.: Graph Transformation for Specification and Programming. In: Science of Computer Programming 34 (1999), August, Nr. 1, p. 1–54

[BKPPT01] Bottoni, P. ; Koch, M. ; Parisi-Presicce, F. ; Taentzer, G.: A Visualization of OCL Using Collaborations. In: Gogolla, M. (Hrsg.) ; Kobryn, C. (Hrsg.): UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference, Toronto, Canada. Heidelberg : Springer Verlag, October 2001 (LNCS 2185). – ISBN 3–540–42667–1

[BKPPT02] Bottoni, P. ; Koch, M. ; Parisi-Presicce, F. ; Taentzer, G.: Working on OCL with Graph Transformation. In: APPLIGRAPH Workshop on Applied Graph Transformation (AGT 2002), Grenoble, France, 2002, p. 1–10

[Budinsky] Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J. Eclipse Modeling Framework: A Developer's Guide Chapter 2. Available at: http://www.awprofessional.com/content/images/0131425420/samplechapter/budinskych02.pdf

[C4ISR] C4ISR Architecture Working Group (1997), C4ISR Architecture Framework Version 2.0, US Department of Defense, Dec. 18, 1997. http://www.c3i.osd.mil/org/cio/i3/AWG_Digital_Library/pdfdocs/fw.pdf

[CH03] Czarnecki, K. ; Helsen, S. Classification of Model Transformation Approaches. The Third OOPSLA Workshop on Domain-Specific Modeling, OOPSLA 2003 Workshop, Anaheim, CA, USA – http://www.softmetaware.com/oopsla2003/czarnecki.pdf. October 2003

[Charlesworth] Charlesworth, I. Application Developments Environments Technology Audit: Interactive Objects ArcStyler. ArcStyler whitepaper. Available at: http://www.io-software.com/as_support/brochures/ArcStyler_TECH_RPS_1098.pdf

[CHM+02] Csertán, G. ; Huszerl, G. ; Majzik, I. ; Pap, Z. ; Pataricza, A. ; Varró., D.: VIATRA - Visual Automated Transformations for Formal Verification of UML Models. In: International Conference on Software Engineering (ASE 2002), Edinburgh, Scotland, IEEE Computer Society, September 2002. – ISBN 0–7695–1736–6, p. 267–270

[Cook jan04] Steve Cook, Domain Specific Modelling and MDA, MDA Journal, Jan 2004.

[DHO01] Demuth, B. ; Hussmann, H. ; Obermaier, S. ; Whittle, J. (Hrsg.). Experiments With XMI Based Transformations of Software Models. WTUML: Workshop on Transformations in UML, ETAPS 2001 Satellite Event, Genova, Italy. April 2001

[Douglass] Douglass, B. P. Model Driven Architecture and Rhapsody. I-Logix Rhasody whitepaper. Available at: http://www.ilogix.com/whitepaper_PDFs/whitepapers.cfm?pdffile=MDAandRhapsody.pdf

[Eclipse] Eclispse Organization. Eclispe.org. Available at: http://www.eclipse.org/

[EEKR99] Ehrig, H. ; Engels, G. ; Kreowski, H.-J. ; Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools. World Scientific Pub Co, October 1999. – ISBN 9–810–24020–1 NoE INTEROP – WP9 Intelligent Networks and Management of Distributed Systems (iVS) Technical University of Berlin 5/7

[EMF Model] Eclispse Organization. Generating an EMF Model.. EMF Document. Available at: http://download.eclipse.org/tools/emf/scripts/docs.php?doc=tutorials/clibmod/clibmod.html

[EMF.Edit] Eclispse Organization. The EMF.Edit Framework Ovierview.. EMF Document. Available at: http://download.eclipse.org/tools/emf/scripts/docs.php?doc=references/overview/EMF.Edit.html

[EMF] Eclispse Organization. The Eclipse Modeling Framework (EMF) Overview. EMF Document. Available at: http://download.eclipse.org/tools/emf/scripts/docs.php?doc=references/overview/EMF.html

[Evaluation OptimalJ] King's College London, University of York. An Evaluation of Computer OptimalJ Professional Edition as an MDA Tool. CompuwareCorportation OptimalJ whitepaper. Available at: http://www.compuware.com/dl/kings_mda.pdf

[FNTZ98] Fischer, T. ; Niere, J. ; Torunski, L. ; Zündorf, A.: Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In: Ehrig, H. (Hrsg.) ; Engels, G. (Hrsg.) ; Kreowski, H.-J. (Hrsg.) ; Rozenberg, G. (Hrsg.): Theory and Application of Graph Transformations, 6th International Workshop (TAGT'98), Paderborn, Germany. Heidelberg : Springer Verlag, November 1998 (LNCS 1764). – ISBN 3–540–67203–6, p. 296–309

[Frankel] D. Frankel. Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley. January 2003. ISBN: 0471319201.

[Gardener dec03] Gardner, T. Model-Driven Metadata Integration using MOF 2.0 and Eclipse. OMG MDA Implementers' Workshop. Available at: http://www.omg.org/news/meetings/workshops/MDA_2003-2_Manual/1-3_Gardner.pdf

[GERAM] IFIP-IFAC Task Force (1999), GERAM: Generalised Enterprise Reference Architecture and Methodology, Version 1.6.3, March 1999 (Published also as Annex to ISO WD15704). http://www.fe.up.pt/~jjpf/isf2000/v1_6_3.html

[Gery, Harel, Palachi] Gery, E., Harel, D., and Palachi, E. Rhapsody: A Complete Life-Cycle Model-Based Development System. I-Logix Rhasody whitepaper. Available at: http://www.ilogix.com/whitepaper_PDFs/Rhapsody-ifm.pdf

[GJ90] Garey, M. R. ; Johnson, D. S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman Company, November 1990. – ISBN 0–716–71045–5

[Greefield, Short] J. Greenfield and K. Short. Software Factories: Assembling Applications with Patterns, Frameworks, Models and Tools. John Wiley and Sons. 2004.

[HE00] Heckel, R. ; Engels, G.: Graph Transformation and Visual Modeling Techniques. In: BEATCS 71 (2000), June, p. 186–203

[HJGP99] Ho, W. M. ; Jézéquel, J.-M. ; Guennec, A. ; Pennaneac'h, F.: UMLAUT: An Extendible UML Transformation Framework. In: 14th International Conference on Automated Software Engineering (ASE 99), Florida, US, IEEE Computer Society, 1999. – ISBN 0–7695–0415–9, p. 275–278

[IO-ArcStyler] Interactive Objects ArcStyler Product Tour. ArcStyler Documentation. Available at: http://www.io-software.com/products/arcstyler_product_tour.jsp

[IO-Cartridges] Interactive Objects Extensible MDA-Cartridges. ArcStyler Documentation. Available at: http://www.io-software.com/products/arcstyler_cartridges.jsp

[IO-Software] Interactive Objects ArcStyler: Leading the Way in Model Driven Architecture. ArcStyler Documentation. Available at: http://www.io-software.com/products/arcstyler_overview.jsp

[ISO/IEC 10746-1] Open Distributed Processing - Reference Model - Part 1: Overview, ITU Recommendation X.901 | ISO/IEC 10746-1, International Telecommunication Union, 1998.

[ISO/IEC 10746-2] Open Distributed Processing - Reference Model - Part 2: Foundations, ITU Recommendation X.902 | ISO/IEC 10746-2, International Telecommunication Union, 1996.

[ISO/IEC 10746- Open Distributed Processing - Reference Model - Part 3: Architecture, ITU Recommendation X.903 | ISO/IEC

3] 10746-3, International Telecommunication Union, 1996.

[ISO/IEC 10746-4] Open Distributed Processing - Reference Model - Part 4: Architectural Semantics, ITU Recommendation X.904 | ISO/IEC 10746-4, International Telecommunication Union, 1998.

[Jon01] Jones, S. P.: Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell. In: Engineering theories of software construction (2001), p. 47–96. ISBN 1–586–03172–4

[KH02] Kovse, J. ; Harder, T.: Generic XMI-Based UML Model Transformations. In: Bruel, J.-M. (Hrsg.) ; Bellahséne, Z. (Hrsg.): Int. Conf. on Object-Oriented Information Systems (OOIS'02), Montpellier, France. Heidelberg : Springer Verlag, September 2002 (LNCS 2426), p. 192–198

[Kleppe, Warmer, Bast] Anneke Kleppe, Jos Warmer, Wim Bast. MDA Explained: The Model Driven Architecture--Practice and Promise. Addison-Wesley Professional; 1st edition. April 2003. ISBN: 032119442X.

[KTW02] Kiesner, C. ; Taentzer, G. ; Winkelmann, J.: A Visual Notation of the Object Constraint Language / TU-Berlin. 2002 (No. 2002/23). – technical report

[LB93] Löwe, M. ; Beyer, M.: AGG - An Implementation of Algebraic Graph Rewriting. In: Rewriting Techniques and Applications, 5th International Conference (RTA-93), Montreal, Canada. Heidelberg : Springer Verlag, June 1993 (LNCS 690). – ISBN 3–540–56868–9, p. 451–456

[MDA Journal] The MDA Journal. http://www.davidfrankelconsulting.com/MDAJournal.htm.

[MDA Manifesto] G. Booch, A. Brown, S. Iyengar, J. Rumbaugh, B. Selic, "An MDA Manifesto," In D. Frankel (ed.), MDA Journal, Business Process Trends, May 2004

[Mellor etal] Stephen J. Mellor, Kendall Scott, Axel Uhl, Dirk Weise. MDA Distilled (Addison-Wesley Object Technology Series). Addison-Wesley Professional. March 2004. ISBN: 0201788918.

[Modelbased] Modelbased.Net MDA Tools. Available at: http://www.modelbased.net/mda_tools.html

[Obj03] Objects, Interactive. ArcStyler. http://www.io-software.com/. November 2003

[OMG MDA] OMG, MDA Guide Version 1.0.1, Joaquin Miller and Jishnu Mukerji. (eds.), 2003, www.omg.org.

[OMG03d] OMG: UML 2.0 Infrastructure Specification / Object Management Group.2003. – OMG formal document 03-09-15

[OMG03i] OMG: XML Metadata Interchange (XMI) Specification / Object Management Group. 2003. – OMG formal document 03-05-02

[OptimalJ MDA] CompuwareCorporation. Compuware OptimalJ standardizeds on Object Management Group's Model Driven Architecture. CompuwareCorportation Documentation. Available at: http://www.compuware.com/products/optimalj/1812_ENG_HTML.htm

[OptimalJ Product Preview] CompuwareCorporation. OptimalJ Product Preview. CompuwareCorportation Documentation. Available at: http://www.compuware.com/products/optimalj/1821_eng_html.htm

[PBG01] Peltier, M. ; Bezivin, J. ; Guillaume, G. ; Whittle, J. (Hrsg.). MTRANS, a general framework based on XSLT, for model transformations. WTUML: NoE INTEROP – WP9 Intelligent Networks and Management of Distributed Systems (iVS) Technical University of Berlin 6/7 Workshop on Transformations in UML, ETAPS 2001 Satellite Event, Genova,Italy. April 2001

[Pyt03] Python. The Python Homepage. http://www.python.org. November 2003

[PZB00] Peltier, M. ; Ziserman, F. ; Bezivin, J. On levels of model transformation. XML Europe 2000, Paris, France –

[Rhapsody] I-Logix. Rhapsody: Model-Driven Development with UML 2.0 and Beyond. I-Logix Rhapsody Documentation. Available at: http://www.ilogix.com/rhapsody/rhapsody.cfm

[Sar02] Sarkar, S.: Model-Driven Programming using XSLT. In: XML Journal 3 (2002), August, Nr. 8, p. 42–51

[SPH+01] Suny´e, G. ; Pennaneac'h, F. ; Ho, W. M. ; Guennec, A. ; Jézéquel, J.-M.:Using UML Action Semantics for executable modeling and beyond. In: Dittrich, K. R. (Hrsg.) ; Geppert, A. (Hrsg.) ; Norrie, M. C. (Hrsg.): Advanced Information Systems Engineering (CAiSE 2001), Interlaken,Switzerland. Heidelberg : Springer

Verlag, June 2001 (LNCS 2068). – ISBN 3–540–42215–3, p. 433–447

[Tae99] Taentzer, G.: AGG: A Tool Environment for Algebraic Graph Transformation. In: Nagl, M. (Hrsg.) ; Schürr, A. (Hrsg.) ; Münch, M. (Hrsg.): Applications of Graph Transformations with Industrial Relevance, International Workshop,AGTIVE'99, Kerkrade, The Netherlands. Heidelberg : Springer Verlag, September 1999 (LNCS 1779). – ISBN 3–540–67658–9, p. 481–488

[TOGAF] The Open Group (2002), The Open Group Architectural Framework (TOGAF) Version 8 "Enterprise Edition". The Open Group, Reading, UK. http://www.opengroup.org/togaf/.

[Tri01] Triskel Project, IRISA. UMLAUT: Unified Modeling Language All pUrposes Transformer. URL http://www.irisa.fr/UMLAUT. 2001

[UMT] Oldevik, J. UML Model Transformation Tool: Overview and user guide documentation. UMT documentation. Available at: http://umt-qvt.sourceforge.net/docs/UMT_documentation_v08.pdf

[UMT-QVT] Modelbased.Net UMT-QVT Homepage. UMT documentation. Available at: http://umt-qvt.sourceforge.net/

[VGP01] Varró, D. ; Gyapay, S. ; Pataricza, A. ; Whittle, J. (Hrsg.). Automatic Transformation of UML Models for System Verification. WTUML: Workshop on Transformations in UML, ETAPS 2001 Satellite Event, Genova, Italy. April 2001

[VVP02] Varró, D. ; Varró, G. ; Pataricza, A.: Designing the Automatic Transformation of Visual Languages. In: Science of Computer Programming 44(2002), August, Nr. 2, p. 205–227

[What is EMF] JISC National Mirror Service. What is EMF? Availabe at: http://text.mirror.ac.uk/mirror/download.eclipse.org/tools/emf/scripts/home.php

[Zachman 87] Zachman, J.A. (1987), A Framework for Information Systems Architecture, IBM Systems Journal, 26(3):276–292.

[Zachman 92] Sowa J.F., Zachman J.A. (1992), Extending and Formalizing the Framework for Information Systems Architecture, IBM Systems Journal, 31(3): 590–616.

[Zee, Laagland, Hafkenscheid] Zee, H. van der, Laagland, P. and Hafkenscheid, B. (eds.) (2000), Architectuur als Management Instrument - Beheersing en Besturing van Complexiteit in het Netwerktijdperk. (in Dutch).

Metamodelling

- [Atkinson and Kühne 2003] C. Atkinson and T. Kühne, "Model-Driven Development: A Metamodeling Foundation", IEEE Software, vol. 20, no. 5, pp. 36-41, 2003. http://www.mm.informatik.tu-darmstadt.de/staff/kuehne/publications/papers/mda-foundation.pdf
- [Clark, et al. 2004] T. Clark, A. Evans, P. Sammut, and J. Willans, "Applied Metamodelling - A Foundation for Language Driven Development, Version 0.1", 2004. http://albini.xactium.com/web/index.php?option=com_remository&Itemid=54&func=select&id=1
- [Seidewitz 2003] E. Seidewitz, "What Models Mean", IEEE Software, vol. 20, no. 5, pp. 26-32, 2003.
- [Swithinbank, et al. 2005] P. Swithinbank, M. Chessell, T. Gardner, C. Griffin, J. Man, H. Wylie, and L. Yusuf, "Patterns: Model-Driven Development Using IBM Rational Software Architect", IBM, Redbooks, December 2005. http://www.redbooks.ibm.com/redbooks/pdfs/sg247105.pdf

Language engineering

- [Atkinson and Kühne 2003] C. Atkinson and T. Kühne, "Model-Driven Development: A Metamodeling Foundation", IEEE Software, vol. 20, no. 5, pp. 36-41, 2003.

http://www.mm.informatik.tu-darmstadt.de/staff/kuehne/publications/papers/mda-foundation.pdf

- [Clark, et al. 2004] T. Clark, A. Evans, P. Sammut, and J. Willans, "Applied Metamodelling - A Foundation for Language Driven Development, Version 0.1", 2004. http://albini.xactium.com/web/index.php?option=com_remository&Itemid=54&func=select&id=1
- [Seidewitz 2003] E. Seidewitz, "What Models Mean", IEEE Software, vol. 20, no. 5, pp. 26-32, 2003.
- [Swithinbank, et al. 2005] P. Swithinbank, M. Chessell, T. Gardner, C. Griffin, J. Man, H. Wylie, and L. Yusuf, "Patterns: Model-Driven Development Using IBM Rational Software Architect", IBM, Redbooks, December 2005. http://www.redbooks.ibm.com/redbooks/pdfs/sg247105.pdf

Eclipse technologies

- EMF, http://www.eclipse.org/emf
- Help and tutorials, http://help.eclipse.org
- GEF, http://www.eclipse.org/gef
- GMF, http://www.eclipse.org/gmf
- GMF Tutorials
    - Cheat Sheats in Eclipse Help->Cheat Sheats
    - http://wiki.eclipse.org/index.php/GMF_Tutorial

Model transformations

- OMG, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification", Object Management Group (OMG), Document ptc/05-11-01, November 2005. http://www.omg.org/docs/ptc/05-11-01.pdf
- INRIA, "ATL - The Atlas Transformation Language Home Page". http://www.sciences.univ-nantes.fr/lina/atl/
- Eclipse.org, "ATL Home page". http://www.eclipse.org/gmt/atl/
- IBM, "Model Transformation Framework". http://www.alphaworks.ibm.com/tech/mtf/
- IBM, "Model Transformation with the IBM Model Transformation Framework". http://www-128.ibm.com/developerworks/rational/library/05/503_sebas/