



I T E A

INFORMATION TECHNOLOGY
FOR EUROPEAN ADVANCEMENT

MARTES

**Model-based Approach for Real-Time Embedded
Systems development**

Specification of the Model Driven Engineering Process

Deliverable ID: 1.7
Version: 1.0
Date: 14/06/2007
Editor: Stefan Van Baelen
Status: Final
Confidentiality: Public

MARTES	Specification of the Model Driven Engineering Process	Page : 2 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

This document is part of the work of the EUREKA Σ! 2023 – ITEA 04006 project MARTES.
Copyright © 2005-2006-2007 MARTES consortium.

Authors:

Author	Partner	Email
Stefan Van Baelen	K.U.Leuven	Stefan.VanBaelen@cs.kuleuven.be
Bert Vanhooff	K.U.Leuven	Bert.Vanhooff@cs.kuleuven.be
Johan Devos	Barco	Johan.R.Devos@barco.com
Kari Tiensyrjä	VTT	kari.tiensyrja@vtt.fi
Martin Höst	Lund University	martin.host@telecom.lth.se
Thierry Saunier	Thales DLJ	Thierry.Saunier@fr.thalesgroup.com
Fernando López	TI+D	fla@tid.es

Document History:

Date	Version	Editor	Description
31/03/07	0.1	Stefan Van Baelen	Initial draft based on D1.4
14/06/07	1.0	Stefan Van Baelen	Inputs from K.U.Leuven and TI+D

Filename: MARTES_D1.7_v1.0.doc

MARTES	Specification of the Model Driven Engineering Process	Page : 3 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

TABLE OF CONTENTS

Executive Summary	6
1 Introduction	7
2 Application Development versus MDD Environment Development	8
2.1 Introduction	8
2.2 Developing an MDD Environment	9
2.3 Conclusions	16
3 The MARTES MDD Process Framework	18
3.1 MARTES Process Requirements	18
3.2 MDDE Development Case	18
3.3 Towards a MARTES MDDE Process Framework	19
3.4 Example of MARTES Process descriptions	24
4 MDE Traceability	29
4.1 What is Traceability?.....	29
4.2 Classifying Traceability	30
4.3 Traceability Representation	31
4.4 Traceability Interchange	34
4.5 Choosing the Right Technique	34
4.6 Traceability as Input for Model Transformations	35
5 MDE and Standardization	43
5.1 DO-178B	43
5.2 SPEM.....	45
6 Conclusions	49
7 Acronyms	51
8 References	52

MARTES	Specification of the Model Driven Engineering Process	Page : 4 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

LIST OF FIGURES

Figure 1. Relative effort required to build an application and the supporting environment (larger area indicates larger required effort).	9
Figure 2. Development of an MDDE as a separate process.	10
Figure 3. Relative effort for each discipline. The white dotted lines give a more realistic situation.	12
Figure 4. Focus shift of disciplines in an MDD approach. Original proportions are drawn as full lines.	13
Figure 5. Interweaving two processes facilitates early feedback.	14
Figure 6. The MDDE development case for MARTES.	19
Figure 7. The MARTES Y-Chart MDDE.....	20
Figure 8. The Y-Chart MDDE usage process	21
Figure 9. The complete common MARTES process framework.....	21
Figure 10. The MDDE refinement process phases.	22
Figure 11. Requirement Analysis activity diagram.....	22
Figure 12. Requirement Allocation activity diagram.....	23
Figure 13. Y-chart refinement activity diagram.	23
Figure 14. MethodContentPackage DefineUsageScenarios.....	25
Figure 15. MethodContentPackage ModelUseCases	25
Figure 16. MethodContentPackage RefineUseCases.....	26
Figure 17. MethodContentPackage ValidateUseCases	26
Figure 18. MethodContentPackage CreateSpecifications.....	26
Figure 19. Requirements and Specification Modeling with Use Cases.....	27
Figure 20. Summary of UML to SystemC translation process.	28
Figure 21. Traceability links, shown as dotted arrows	29
Figure 22. Copy attributes and add an accessor – variation 1.....	31
Figure 23. Copy attributes and add an accessor – variation 2.....	31
Figure 24. Explicitly creating traces.....	31
Figure 25. Two extreme traceability metamodels.	33
Figure 26. Deriving model relations from past traceability information.	36
Figure 27. Using traceability in a transformation chain to generate a relational persistence layer.	38
Figure 28. Navigating tagged trace links.	39
Figure 29. The conceptual SPEM model.	46
Figure 30. Method Content definition versus the application of Method Content in a Process.	48

MARTES	Specification of the Model Driven Engineering Process	Page : 5 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

Figure 31. Key terminology defined in this specification mapped to Method Content versus Process. 48

MARTES	Specification of the Model Driven Engineering Process	Page : 6 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

Executive Summary

This document constitutes the MARTES project deliverable D1.7 “Specification of the MARTES Model Driven Engineering Process” final document, which is an updated and extended version of version 1 contained in the deliverable D1.4.

The objective is to specify the MARTES Model Driven Engineering (MDE) Process, which complements the MARTES models (defined in Deliverable D1.2 & D1.5) and MARTES model transformations (defined in Deliverable D1.3 & D1.6). Since an MDE process must be tailorable for large industrial multi-site applications developments, this document does not specify a single specific MARTES process, but describes a process framework that can be used to instantiate a specific process according to the environment in which the process will be applied.

Chapter 1 summarizes the goals and expected properties of the MARTES MDE Process.

Chapter 2 points out the difference between the development of an application and the development of the environment that facilitates the former, all in the context of MDD. It is explained what exactly an MDD Environment is and its importance in MDD oriented development is emphasized. Furthermore, we discuss the classical Unified Process – a commonly used software development process – and explain how its general principles can be applied for the development of an MDD Environment.

Chapter 3 situates the MARTES MDD Process Framework as a part of the complete MARTES MDD Environment. Following the MDDE development process described in Chapter 2, an initial MDDE for MARTES is presented. First we restate the most important requirements, which we take as input to describe the MARTES MDDE Development Case. The MARTES MDDE Process Framework is defined, and its application is illustrated by presenting a number of MARTES Example Processes.

Chapter 4 deals with traceability, which is needed in MDD in order to establish relationships between the different work products in an MDD process. We first explain what we define as traceable and then classify different traceability approaches along two axes:

- Requirements versus Transformation Traceability,
- Generic/Full versus Specific Traceability.

We continue by discussing two possibilities to represent traceability information and how to make that information interchangeable between different partners. Finally we make a selection of what we think is the best combination of techniques to use for MARTES.

Chapter 5 presents some standards that are of influence on the definition of the MARTES MDD Process Framework, namely the DO-178B standard, the standard for the development of airborne software in the civil aviation industry, and OMG's SPEM standard, the Software Process Engineering Meta-Model that defines concepts for modeling software development processes.

The MARTES MDD Environment and its MARTES MDD Process Framework serves as a base for the definition of a suitable partner- or domain- specific Model Driven Engineering Process, including the development of adequate supporting MDD tools.

MARTES	Specification of the Model Driven Engineering Process	Page : 7 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

1 Introduction

The purpose of this document is to specify the MARTES Model Driven Engineering Process. This process complements the MARTES models, which are defined in Deliverable D1.2 (initial) and D1.5 (final) "Specification of the Models (PIM/PSM) in the MARTES methodology", and MARTES model transformations, which are defined in Deliverable D1.3 (initial) and D1.6 (final) "Specification of the Model Transformations in the MARTES methodology".

Model-driven design requires a strong process to support the design flow and efficient usage of the toolset. MDA-based lifecycle processes have to support efficient process improvement methodologies. The DO-178B standard, specified for the certification of airborne software in the civil aviation industry, will be used as a starting reference and expanded for other application domains (telecom, mobile, residential).

A Model-Driven Engineering (MDE) process is described, as an update of current design processes and tailorable for large industrial multi-site applications developments. This will allow iterative improvements of the engineering efficiency, through configuration management, traceability of information/data, and the definition of an estimation process (concurrency, synchronization, data transfer and storage, cost, performance; e.g. latency, throughput, power consumption...). Those quality attributes, extracted at the different abstraction levels will be integrated into the abstract modeling framework throughout the hierarchy, maintainable through requirements management tools and specified through a set of guidelines and checklists.

Capturing, managing and tracing the architectural knowledge are also an essential part of the design. Its encapsulation into UML notations will ensure the consistency, reversibility and modularity of the process. An easy maintenance for later reuse will then be performed, and stored in a service model repository database, which will be put in place. In addition a quality management process must provide heuristics for repartitioning and reconfiguring (traceability, manageability & predictability metrics).

This document does in fact not specify a single specific MARTES process, but rather describes a process framework that can be used to instantiate a specific process according to the environment in which the process will be applied.

MARTES	Specification of the Model Driven Engineering Process	Page : 8 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

2 Application Development versus MDD Environment Development

In this part we point out the difference between the development of an application and the development of the environment that facilitates the former, all in the context of MDD. In Section 2.1 we explain what exactly an MDD Environment is and emphasize its importance in MDD oriented development. In Section 2.2, we discuss the classical Unified Process – a commonly used software development process – and explain how its general principles can be applied for the development of an MDD Environment. In Section 2.3, we summarize the main elements within an MDD Environment.

2.1 Introduction

Since a few years you would have to be blind not to see the terms MDD, MDE or MDA appear everywhere. After a quest to find out how these techniques can be applied on your project, you probably found out that there is no such thing as "THE" MDD/MDE/MDA. It is rather a collection of very different practices and techniques that have only one thing in common: the use of (abstract) models and transformations that enable the generation of (parts of) an application. Put in other words, all existing and successful MDD approaches are very specifically tailored to a certain project or to a certain type of application. There is no such thing as a generically applicable MDD way of developing. MDD in general can merely be seen as a collection of good practices related to modeling and a bunch of standards and tools that can help to introduce a more model-oriented way of developing software. It is completely up to the developers to select and define the right elements and combine them in a useful and meaningful way. Amongst the elements that must be defined are metamodels, concrete modeling languages and transformations. But, also the technology in which to express these and the tools that support model transformations can differ and must be chosen.

Even selecting predefined elements such as the well-known Unified Modeling Language (UML) does not completely relieve the burden of defining MDD elements. This is a consequence of the fact that the UML (1) is probably too big and too complicated to use as a whole, (2) may not be a *perfect* fit for your needs (needs customization) and (3) is deliberately defined with certain points of ambiguity to enable wide reuse (the so called *semantic variation points*, which need to be resolved before any use).

We refer to the collection of all the necessary elements (tools, metamodels, languages, transformations, usage methods, etc.) that are needed to support developers in applying MDD to a specific project as a *Model Driven Development Environment (MDDE)*.

An MDDE provides the main support for developing applications according to MDD principles. In classical software development, the role of development environment is usually played by a readily available Integrated Development Environment (IDE) or, in the more primitive case, a combination of text editor, compiler, debugger, etc. All the classical development environments have in common that they are centered around concrete programming language and are therefore also very reusable.

Figure 1 shows that, typically, a classical software development environment offers less support for building applications because it is more generic but requires little effort to set up.

MARTES	Specification of the Model Driven Engineering Process	Page : 9 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

On the contrary, an MDD Environment requires a lot more effort to build but offers much more application development support since it is specifically created for one specific application or set of similar applications, i.e. in a product line. In contrast to classical software development we need to put energy into the development of meta-models, specific notations, transformations between different knowledge domains, etc. All these activities require many different skills and in the case of transformations bridge more areas of expertise. Mind that an MDDE does not have to be build from the ground up every time. It is important to capitalize on past experience when building a new MDDE, often for a similar application. It is therefore important to always keep in mind the reusability of the artifacts, such as transformations.

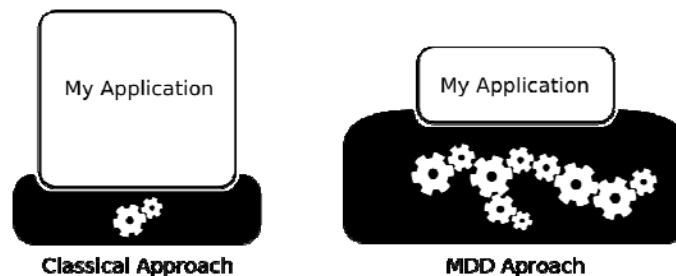


Figure 1. Relative effort required to build an application and the supporting environment (larger area indicates larger required effort).

Due to the need for many different areas of expertise and the profound influence of the MDDE on further application development, the creation of an MDDE is an extremely difficult and important task and should be executed with great care. Because of this complexity and importance we believe that it is beneficial to define a process that guides us in the development of MDD Environments. Such a process would logically be executed before the actual application development can commence however, we will take a slightly different approach.

2.2 Developing an MDD Environment

In section 2.1 we defined an MDD Environment as the collection of all elements that support the development of an application or a number of applications (in a product line) in an MDD context. To be more precise we enumerate a more complete list of elements that encompass an MDD Environment:

1. Infrastructure content
(This is the part of the MDDE that will run-time support, is most cases as a set of cooperating tools)
 - One or more modeling platforms (roughly equivalent to a metamodels) that formalize the abstract syntax of the system parts that need to be modeled. These directly determine the abstraction levels that will be applied.
 - Appropriate modeling languages, or concrete syntax, for each modeling platform.
 - Transformation specifications that specify how to transform models from one modeling platform to another or to code.

MARTES	Specification of the Model Driven Engineering Process	Page : 10 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

- A chain of transformation executions that relate all used models. This chain gives a complete overview how transformations and models are used throughout the development process.
- The necessary tools that support the former elements, together with required customizations. In the common case these will be: a metamodel repository, a drawing tool, a transformation engine and a transformation chaining tool.

2. Method content

- Methods that describe how the infrastructure has to be used. The *method content* (as it is called in 5.2.2) needs to be incorporated into the development processes of applications that are build using the MDDE. This part of the MDDE does not contain run-time support; it is the collection of roles, templates, tasks, etc. that are appropriate to guide the use of the MDDE.

Because of the profound influence of an MDD Environment on the development of an actual application, it is of utmost importance to consider each of its elements in great detail. Developing an MDDE is a very complex task and we cannot just rely on “experts” to define it for us since the required expertise is so diverse and extensive and these experts simply do not exist yet. Therefore we think that the development of an MDDE needs to be guided by a development process of its own.

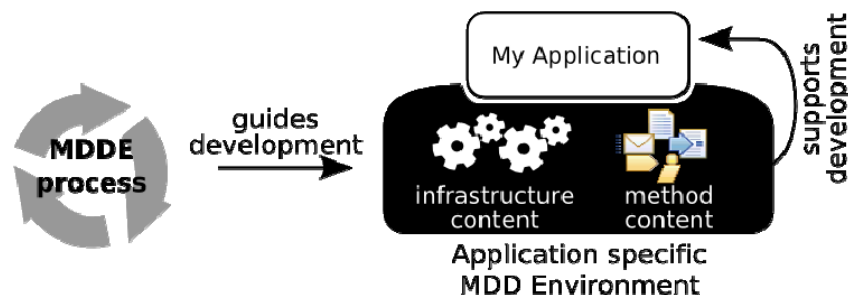


Figure 2. Development of an MDDE as a separate process.

Figure 2 shows the *MDDE process* on the left, which guides the development of a specific MDDE itself. This includes tool support *infrastructure content* (indicated by the cog wheel symbols) and process support in the form of *method content* (indicated by SPEM 2 symbols).

Because the application-types of the different MARTES partners are so different, we think it is useful to put some effort into the definition of a generic MDDE process. Such a generic process can then be used to develop a basic common MARTES MDDE in a structured fashion. A generic MDDE process can also provide common guidelines for each partner to define his own specialized MDD Environment, including a specifically tailored application development process.

In order to provide some context for defining an MDDE process, we introduce one of the most widespread software development processes in the next section: the Unified Process (UP).

MARTES	Specification of the Model Driven Engineering Process	Page : 11 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

2.2.1 Classical Software Development Processes and MDD

Because we treat the development of an MDDE as a software project of its own, we could use, for example, the Unified Process [i,ii, iii] as the software development process for the MDDE. The UP is in fact a generic framework of guidelines and best practices that characterize modern software development. In practice, this framework always needs to be tailored to the needs of a specific project by selecting only the relevant parts and customizing some other parts from the complete UP framework; a tailored instance of the UP is called a *development case*. The general characteristics that each UP-based process possesses are:

- **Iterative and incremental development** The approach splits the development effort into several subsequent fixed-length mini-projects or iterations. The outcome of each of these iterations is an incomplete but working system, whatever that means at each stage. Each iteration includes its own requirements, analysis, design, implementation and testing activities. Each iteration extends the system by tackling new requirements and addresses the feedback from the previous iteration in order to improve the current system.
- **Requirements management and early risk assessment** The goal of the application is captured in use cases, effectively expressing a large subset of the requirements. In each iteration (mini-project), a selected subset of the requirements is addressed and there is an ongoing search for new and changed requirements. The iterative nature facilitates rapid feedback and adaptation. Risks are identified and categorized as early as possible. The requirements that are related to the highest risk and highest importance are tackled in the first iterations.
- **Visual modeling** The UML is used in an ad hoc way to document analysis, design and to improve communication among stakeholders.
- **Modular architectures and reuse** The global structure of a system is emphasized over implementation details since this structure will determine the qualities of the system. There is a preference for reusing existing components over developing everything from scratch.

The UP divides the work in a project into categories, called *disciplines*. A discipline is just a package of *activities* (or work descriptions) and *work products* (or artefacts or deliverables) that are related to a common theme. Typical UP disciplines are Business modeling, Requirements, Design, Implementation, etc. A more common list of disciplines can be found on the left of Figure 3.

As mentioned earlier, a UP project is cut down in relatively small iterations, each addressing a subset of the requirements and each refining the solution until the application is finished. These iterations take a prominent place in the UP and are distinguished by different proportional emphasis on the disciplines (see Figure 3). The figure shows, for example, that the requirements discipline is more important in the beginning of the project and less so near the end.

MARTES	Specification of the Model Driven Engineering Process	Page : 12 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

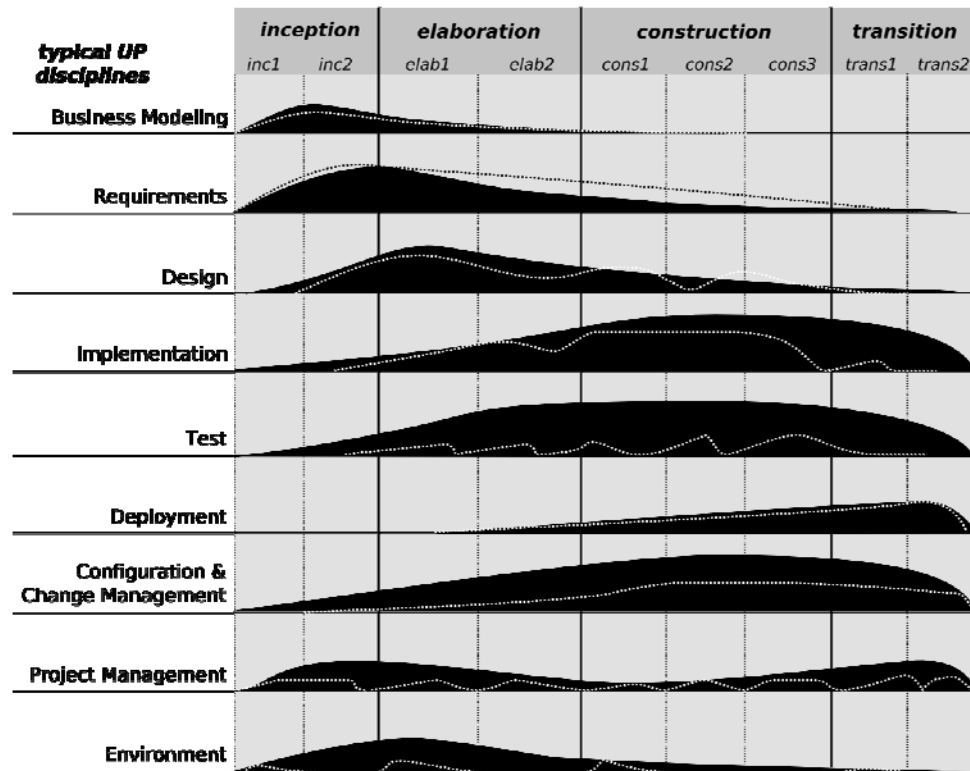


Figure 3. Relative effort for each discipline.
The white dotted lines give a more realistic situation.

We can distinguish different phases in the UP according to the degree of completion of the different discipline work products. Each phase contains its own iterations and is delimited by a so-called milestone. The UP phases are (some of the terms mentioned here will be explained further on):

1. **Inception** – The initial steps to establishing a common vision, the business case, unstructured requirements, etc.
2. **Elaboration** – Refinement of the vision, identification of most requirements, implementation of the core system, etc.
3. **Construction** – Further iterative implementation of remaining requirements, preparation for deployment, etc.
4. **Transition** – Public tests, deployment, etc.

In classical software development practice, the focus is put on the development of the application in terms of gathering requirements, modeling the business domain, writing code and tests, etc. while the means to do this, referred to as the development environment, is often treated as a second class citizen. The obvious reason for this is that classical software development (1) can do without very advanced tools and (2) can rely on a limited set of generic and ready-to-use tools (Integrated Development Environments - IDEs). The UP does address the setup of a development environment up to some point in its *environment discipline* (see bottom of Figure 3). This specific discipline however focuses mainly on the

MARTES	Specification of the Model Driven Engineering Process	Page : 13 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

preparation and configuration of the process itself and less on the development environment in terms of specific tool support.

In the case of MDD, precisely this environment discipline should contain the activities to build an MDDE. But, since that particular discipline is neglected in classical UP, we need to re-evaluate the proportions between the different disciplines and define the environment discipline better if we want to use the UP in an MDD context. Figure 4 shows the shift in proportion of disciplines that we think is appropriate for an MDD approach. The Design discipline becomes more important for a longer portion of the process while the implementation and test disciplines get postponed to later on in the process. Secondly, the environment discipline is much more emphasized.

If we pursue an MDD approach we prefer modeling over code, which means that designing the system becomes more important than coding the system; more detailed models are created, hence requiring more effort in the design/business modelling disciplines. For the real-time embedded domain, the design becomes even more important since many additional, often timing-related, issues are best incorporated at this stage. Many parts of the code can be auto-generated from the models so the implementation effort will be limited to modifying the generated code, which can be done later on in the process. The same goes for testing: many system tests can be auto-generated, again lowering the relative effort needed for testing.

If we emphasize the design/modeling of a system we need of course the means to create these models and to transform them into code/tests. These tasks are facilitated by an MDDE, as defined in Section 2.2. Instead of introducing a new discipline we choose to augment the existing environment discipline with the creation of an project-specific MDDE. Parts of the total development effort are therefore transferred into the environment discipline (Figure 4), which can be seen as a 'mini' project that provides an MDDE that is used by the other disciplines.

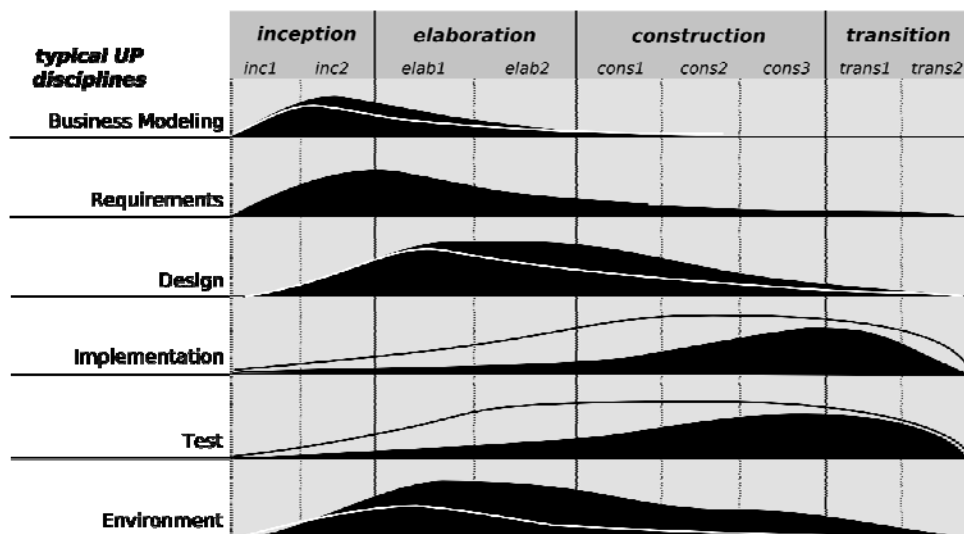


Figure 4. Focus shift of disciplines in an MDD approach. Original proportions are drawn as full lines.

MARTES	Specification of the Model Driven Engineering Process	Page : 14 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

We must notice that the effective amount of work put into the creation of an MDDE can vary tremendously from application to application and from MDD approach to MDD approach. For example, the use of Domain Specific Languages (DSLs) in current practices often comes down to the development of a simple meta-model and notation and a very basic generator that translates to executable component compositions. On the other hand if we choose to model and generate the components themselves, the effort will be considerably larger because the models will have to contain more details. Combinations of these two extremes are also possible, for example modelling components to a certain extent and completing the code manually in combination with DSL to model component compositions.

From the previous, it is clear that a considerable effort is required to build an appropriate MDDE. We propose an MDDE process to guide this effort in the following section.

2.2.2 An Iterative MDDE Development Process

The UP does not give a precise description of the environment discipline, and the philosophy behind it certainly does not take into account any MDDE related issues. Since a big part of application development in an MDD-oriented project can be relocated to the MDD Environment we have emphasized the importance of the environment discipline considerably (Figure 4). The work products for which that discipline is responsible are (1) a tool infrastructure and (2) a set of method elements that guide the use of the tool. Both of them together compose an MDDE, as defined in Section 2.2.

We choose to use the UP as the foundation to build our MDDE process because it is a fairly generic collection of good practices that are applied in many mainstream development processes today (see 2.2.1). Because the development of appropriate MDDEs is still a very young discipline, it is very important that this happens in an incremental fashion with early user feedback. Furthermore it is important to clearly capture specific requirements and assess possible risks. The UP possesses all of these characteristics. Other UP characteristics, such as visual modeling and reuse that were described in the previous section, will also be incorporated in our process.

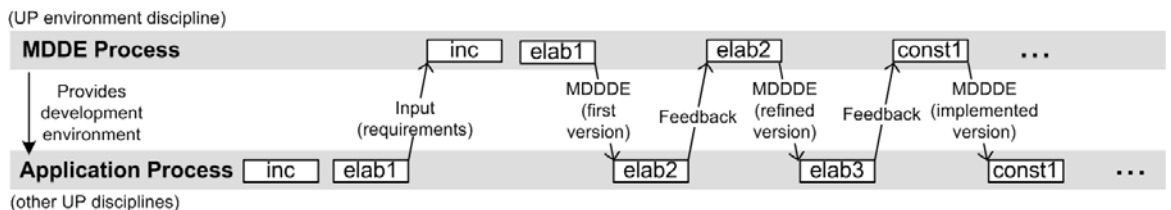


Figure 5. Interweaving two processes facilitates early feedback.

Figure 5 shows a high level overview of the process structure we want to adopt.

We make a clear distinction between the *MDDE process* and the *application process*. The goal of the first is to create an MDDE while the second guides the development of the application itself, supported by the MDDE. It is immediately noticeable from the figure that the two processes are interweaved with one another. We did this deliberately to explicitly indicate that an MDDE can be build and refined incrementally while the normal application process continues. A part of the application process is then dedicated to testing intermediate releases of the MDDE as outcome of the iterations. Because the application process is in

MARTES	Specification of the Model Driven Engineering Process	Page : 15 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

fact the client of the MDDE process it is best suited to give feedback. With this parallel construction it is possible to build an MDDE piece by piece and enrich it with new features when necessary so that the application process can benefit immediately instead of having to wait for a completed MDDE first.

Applying a UP based process for the MDDE process is appropriate because it supports the regular delivery of an incomplete but working system to quickly gather customer (in this case the application process) feedback. Each delivery/feedback loop is indicated by a down and upwards arrow in the figure.

The application process does not have to be UP-based per se (although it is assumed in the figure), as long as it can incorporate the regular feedback loops to the MDDE process. This does implies that the application process is an iterative process. Additionally it needs to be able to easily change the current process configuration and incorporate new method content that is provided by an evolving MDDE.

The first upwards arrow (between *elab1* and *inc*) provides the initial input to the MDDE process and later upwards arrows give feedback after evaluation of intermediate MDDE results (obtained after an iteration of the MDDE process).

In the next section we give an overview of our initial thoughts on the contents of each phase (inception, elaboration, construction and transition) of the MDDE process.

2.2.3 MDDE Process overview

The UP uses a gradual evolution from requirements over conceptual design to implementation. We adopt the same iterative structure as the UP but change the content somewhat to match the specific MDDE needs.

In the *Inception* phase we primarily try to establish a common vision (a set of goals) on the MDDE and identify a basic set of requirements that are applicable to the MDDE. The idea of the MDD Vision (analogously to the classical UP vision document) is to produce a managerial document, identifying the big ideas, the specific problems and what a proposed solution may look like. Part of the initial input for this iteration can come from the output from the first iteration(s) of the application process, when most of the application requirements are gathered. In case of the UP the application requirements are materialized as a vision document, a supplementary specification (non-functional requirements) and a set of use cases (functional requirements).

In summary, the output of this iteration should at least contain the start of the following items:

- A *general vision on MDD* for this specific project with a first proposition of a solution, e.g. coarse-grained abstraction levels, general proposition for a solution structure . This is largely contained in an MDD vision document.
- A list of *application concerns/requirements* that directly or indirectly influence the MDDE. Non-functional requirements are typically found here.
- A list of *constraints*, for example metamodels that must be reused, tools that must be used, implementing technologies that were decided on already, etc.

We aim to use a very structured manner to record requirements because they are part of the communication between the MDDE users (applying the application process) and MDDE

MARTES	Specification of the Model Driven Engineering Process	Page : 16 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

providers (applying the MDDE process). The requirement diagrams provided by SysML in combination with classical use cases seem to be a good candidate.

The **Elaboration** iterations are used to define the logical elements that make up an MDDE (platforms/metamodels, notations, transformations) and work towards a first, partial, implementation. In the first elaboration iteration we need to create rough metamodels (cfr. domain models) for the abstraction levels that we envisioned and informally describe candidate transformations. Domain models, possibly created by the application process during early analysis activities can be used as input. All the artefacts created after the first elaboration iterations will not be implemented, so we need to get feedback from our users via interviewing techniques, etc. at this early stage. Further iterations use this feedback to improve the metamodels if necessary.

The next elaborations can add visual notations to the metamodels so that the users can evaluate the preliminary MDDE more independently. Even later elaborations will focus more on the internal architecture of the MDDE: modularizing transformations, specification of transformations, first implementations, etc.

During the **Construction** iterations, we need to focus on detailed implementation in order to deliver a user-friendly MDDE. At the end of elaboration and beginning of construction the most part of requirements must be addressed in the design and most risks must be tackled.

2.3 Conclusions

We have demonstrated that in MDD-oriented development, a considerable part of the total development effort goes into the development of a suitable Model Driven Development Environment or MDDE. Such an environment provides very specific support to develop the envisioned application by offering an integrated set of modeling languages backed with transformations and specific process support (method content) that guides the use of the environment. An MDDE is similar to an IDE up to a certain point in the sense that both of them provide support to the developers of an application. The difference lies in the specificity of this support: while an IDE provides an application agnostic environment, an MDDE provides a very application-specific environment that can only be used within a very limited scope. Another difference is that an IDE is delivered 'as is' by a tool vendor, while an MDDE is a custom-build development environment that is typically realized by a combination of several different tools (visual editors, transformation engines, etc.) and/or specific tool extensions (own metamodels, custom transformation specifications, etc.).

Building an MDDE requires a considerable amount of effort and requires a large variety of expertise in different domains. Because of this complexity we opted to define a development process that is specifically targeted at developing MDDEs. Our proposition is strongly based on the Unified Process and adheres to its characteristics. We choose for a method that develops the MDDE incrementally in parallel with the application in order to benefit from early results.

In summary, we discussed two processes:

- The *MDDE Process*, which is a generic, UP-based, process. The output of this process is an MDDE that contains an infrastructure part (platforms, languages, transformations, etc. supported by tools) and process part.

MARTES	Specification of the Model Driven Engineering Process	Page : 17 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

- The *Application Process* guides the actual application development and is gradually enriched by the method content that is provided by the MDDE along the way.

This process to define project specific MDDEs seems to be an appropriate first step for MARTES because of the huge variety of partner backgrounds. In this case the process forces a large team to synchronize and deliver certain work products at certain points in time. By separating the knowledge providers/facilitators from the knowledge users and systematically applying a 'provide environment'/'give feedback' loop we can converge to a common solution.

MARTES	Specification of the Model Driven Engineering Process	Page : 18 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

3 The MARTES MDD Process Framework

In this chapter we will follow the MDDE process, that we have described in the previous section, to develop an initial MDDE for MARTES (see also Figure 2). At this point in time it is not feasible to describe a completely worked out MARTES MDDE. Therefore we execute only the initial iterations of the MDDE process and explain how it can interact with and influence the application process of each partner. We refer to the results of these initial steps as the MARTES MDD Process Framework.

In Section 3.1, we restate the most important requirements, which we take as input in Section 3.2 to describe the MARTES MDDE Development Case. In Section 3.3, The MARTES MDD Process Framework is defined. The application of this MARTES MDD Process Framework is illustrated in Section 3.4 by presenting a number of MARTES Example Processes.

3.1 MARTES Process Requirements

The MDD process within the MARTES methodology must have some specific characteristics. We summarize the most important process requirements that are common among the partners here:

- The MDD process must be a generic framework that can be specialized by each partner. It must be flexible enough to allow different management styles and different development cycles (iterative, incremental).
- The MDD process must be lightweight since heavyweight approaches seem to hinder adoption of MDD due to the large efforts required.
- Traceability must be addressed throughout the whole process.
- Use the UML up to its limits but allow DSLs to boost productivity where appropriate
- Foresee reuse at any abstraction level.
- RTES specific requirements:
 - Classical RTES concerns such as timing and resource related requirements must be addressed.
 - The link between application and platform must be very clear.
 - Hardware is directly visible and the allocation of software onto the hardware must be controlled manually.

3.2 MDDE Development Case

As we mentioned in Section 2.2.1, applying the Unified Process means that we first need to define which parts of the UP framework we will use and how we will use it. The same is true for the MDDE process. In short, we need to adapt and tailor the process to our specific needs. Such a tailored process is called a development case according to UP terminology. In the case of the MDDE process we need to concretely define the synchronization of the application process and MDDE process concerning the acquisition of feedback (generic form see Figure 2).

MARTES ITEA 04006	Specification of the Model Driven Engineering Process Deliverable ID: 1.7	Page : 19 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final Confid : Public

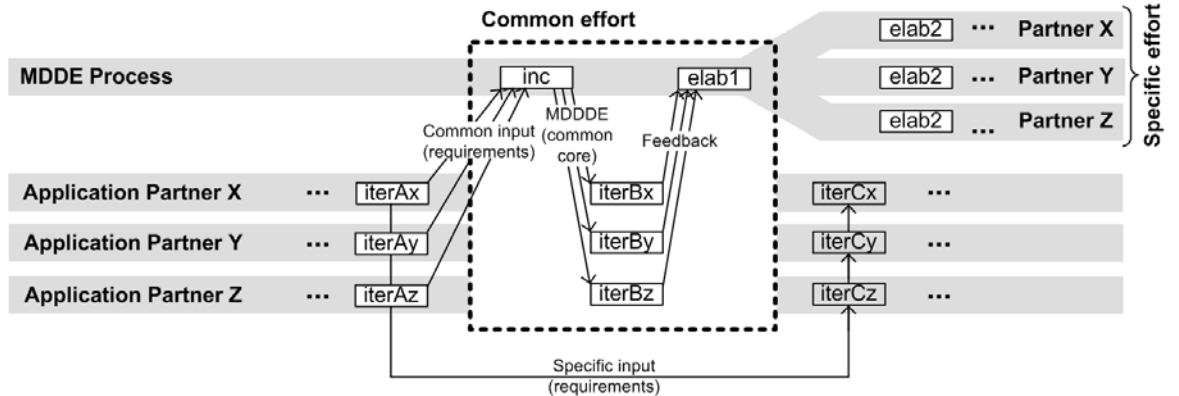


Figure 6. The MDDE development case for MARTES.

In the MARTES case the goal is to define a common process framework that can be further specialized by each partner, according to his unique requirements. Therefore we created a specific development case as shown in Figure 6. We adopt the generic MDDE process, as explained in Section 2.2.3, to start out with (top bar) and assume that the specific application processes of each partner (three bottom bars) are iterative so that a point can be identified where most of the application requirements are determined (*iterAx, y, z*). The union of requirements of the partners' typical applications is the initial input to the inception iteration of the common MDDE development process.

In the inception phase (*inc*) an initial proposal for an MDDE is created. This proposal is then reviewed by each partner in a next iteration or of their respective process (*iterBx, y, z*). The results of this evaluation are used as feedback input for the first elaboration iteration (*elab1*) of the MDDE process. We do not define a structured document format for the feedback at this point because the MDDE now consists only of preliminary; feedback is best discussed during developer meetings.

The output of the first elaboration is an initial common MDDE framework that must be further refined by each partner according to his own specific requirements. This also means that the MDDE process development case that will be followed from this point on will differ for each partner, indicated in Figure 6 by splitting the common MDDE bar into three partner specific bars. Mind that, from hereon, each application process will also be influenced by method content provided by the respective MDDE development case. In the next section we will further refine this MDDE development case to include guidance for the partners on how to specialize the initial MDDE.

3.3 Towards a MARTES MDDE Process Framework

As we consider the MDDE (platforms/metamodels, transformations, method content, etc.) as a whole, it is clear that a number of parts are already defined in the other MARTES deliverables. These parts are mainly focused on the static structure of the common MDDE framework. We quickly review these parts in Section 3.3.1. The process part of the MDDE is discussed in Section 3.3.2. We end this section with some intermediate conclusions in Section 3.3.3.

MARTES	Specification of the Model Driven Engineering Process	Page : 20 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

3.3.1 The Common Y-Chart

In other MARTES deliverables [iv] we already described the common conceptual modeling levels and transformations. A overview of this static view is given in Figure 7.

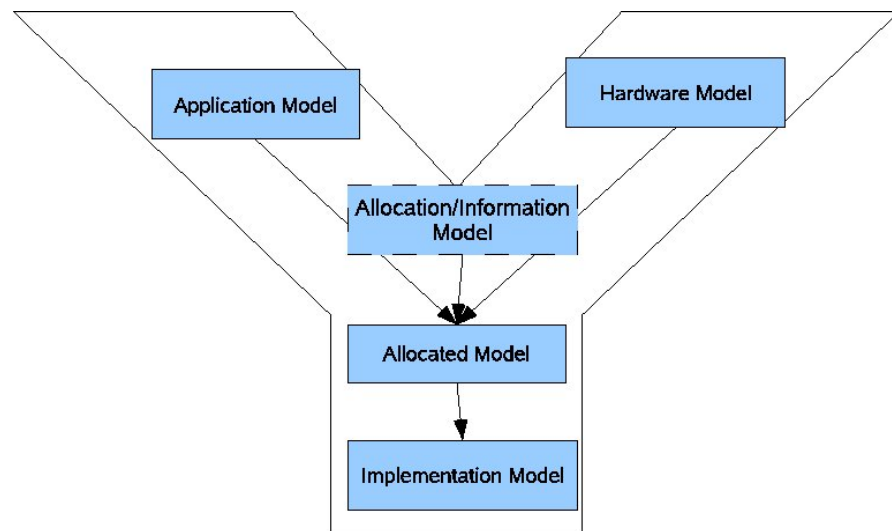


Figure 7. The MARTES Y-Chart MDDE

This Y-chart is considered as the initial proposal for the structure of the common MDDE; it is delivered in the MDD Vision document after one of the first iterations (e.g. *inc* on Figure 6) of the common MDDE process. With this structure we can say something about both the infrastructure content and method content that are part of the MDDE.

It is impossible to go into details at this point but we can at least say that the infrastructure content needs to contain the following elements:

- A set of concrete metamodels that address *real-time aspects* and *platform modelling* with a strong preference for the *UML* in combination with a *RTES profile*;
- Concrete transformations that *combines/weaves* application models with platform models with a preference to heavily modularize into *small transformation steps*;
- A tool to *chain transformations* together easily;
- Implementation technology, preferably the *Eclipse Modeling Framework* and *Atlas Transformation Language*.

The use of the Y-chart somewhat implies a process that guides the way in which the overall infrastructure content must be used (see Figure 8). The phases of this process map straightforward on the model levels given in Figure 7. Because the MDDE can only deliver method content instead of a full process, the process in Figure 8 must be seen as a structure to categorize method content elements. Each of the phases here can contain a number of roles, tasks, and guidance. Each of these method content elements can be incorporated anywhere in the application process, without sticking to the phases in the figure. At this stage the details of the method content cannot be defined yet; for a big part they have to be filled in by each partner individually.

MARTES	Specification of the Model Driven Engineering Process	Page : 21 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

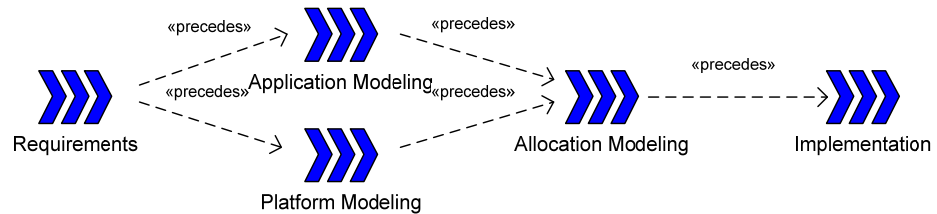


Figure 8. The Y-Chart MDDE usage process

The initial MDDE structure with its infrastructure and method content described in the previous paragraphs obviously still needs refinement. This refinement is accomplished by adding a number of iterations to the MDDE development process in order to establish a complete, partner-specific MDDE. In the following section we discuss how we can execute these subsequent iterations.

3.3.2 The Common MDDE Process Framework

In order to create partner-specific MDDEs based on the Y-chart MDDE, we define a lightweight process that provides guidance to refine the initial MDDE. On the left of Figure 9, indicated with the Y-symbol, we show the common MDDE that we described in Section 3.3.1 and on the right we see a refined version of the MDDE, indicated by a stylized Y. The path to go from the initial MDDE to the refined MDDE is described in the MDDE Refinement Process (in the middle of the figure, using the standard SPEM symbol – see 5.2), that we introduce in this section. This MDDE Refinement Process can be seen as a Y-chart specific development case of the MDDE process. We designate the combination of the common MDDE (Y-chart) and the MDDE Refinement Process as the *MARTES process framework*.

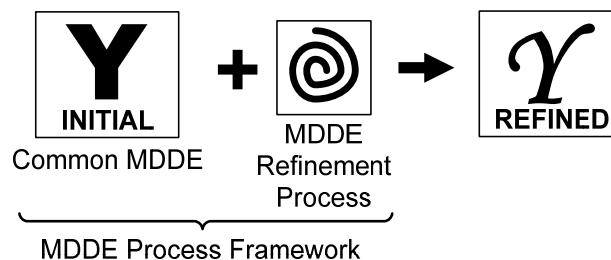


Figure 9. The complete common MARTES process framework.

First, we give an overview of the different processes that we have introduced:

- The *generic MDDE process*, which yielded the initial MDDE Y-chart structure.
- The *application process*, which guides the development of an application and is influenced by the results of the MDDE process through method content.
- The *MDDE refinement process*, which will be elaborated on in the next paragraphs. This process guides the partners in refining the common MDDE to their specific needs.

The MDDE refinement process is split up into four main phases (see Figure 10). The process that we present here is a refinement of the generic MDDE process (presented in Section 2.2), tailored to use the common MDDE (Y-chart). Figure 10 shows, using SPEM 1 notation, that

MARTES	Specification of the Model Driven Engineering Process	Page : 22 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

the first thing we need to do is evaluating the requirements, then refine the common MDDE, design the necessary metamodels and transformations and finally implement them. Each of these phases is further split into smaller and more concrete *work definitions*, which are in turn split into *activities* and *steps* according to SPEM terminology. We will show a number of detailed work definitions in the following paragraphs.

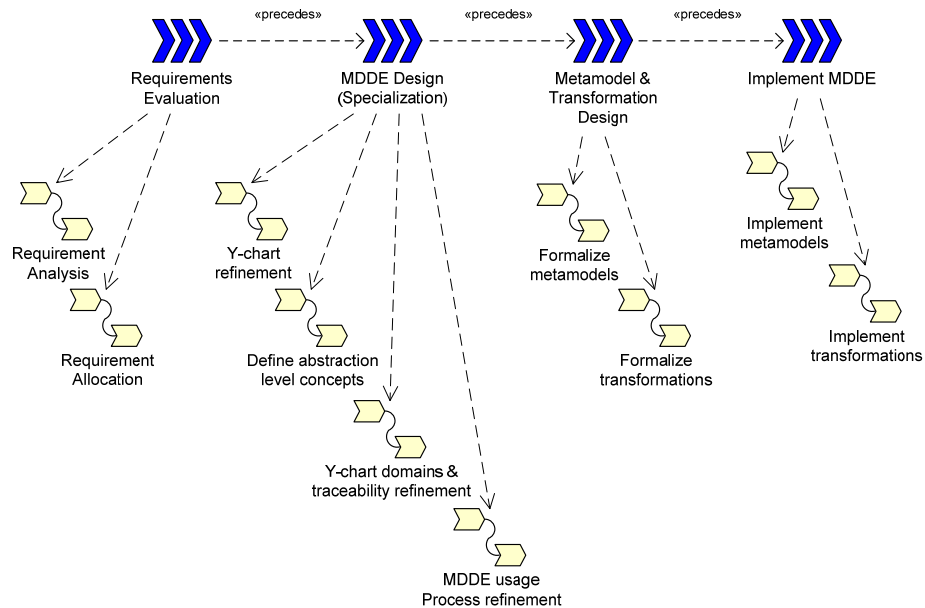


Figure 10. The MDDE refinement process phases.

As shown in Figure 6, the inputs for the MDDE development process mainly come from application specific development processes. At this stage we have not only requirements to take into account, but also the commonly established Y-chart structure. The requirements evaluation phase is therefore responsible to pre-process these requirements and fit them somehow in the Y-chart structure.

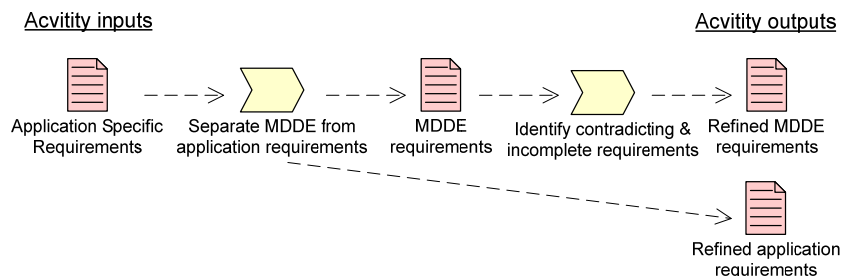


Figure 11. Requirement Analysis activity diagram.

This requirements analysis is split into two work definitions:

- Requirements Analysis, shown in Figure 11. This work definition is a first requirements pre-processing step that (1) separates the MDDE requirements from regular application requirements and (2) identifies contradicting and incomplete requirements. The first activity is needed to decide which parts of the application

MARTES	Specification of the Model Driven Engineering Process	Page : 23 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

development will be assisted by the refined MDDE (Y-chart). The second step explicitly identifies potential problems in the MDDE requirements that need to be addressed in later iterations.

- Requirements allocation, shown in Figure 12. The purpose of this work definition is to separate application (software) related requirements – left side of the Y-chart – from execution platform (hardware & software) related requirements – right side from the Y-chart. These requirements can subsequently be fitted onto the common Y-chart structure, which yields the *MDDE allocated requirement model*.

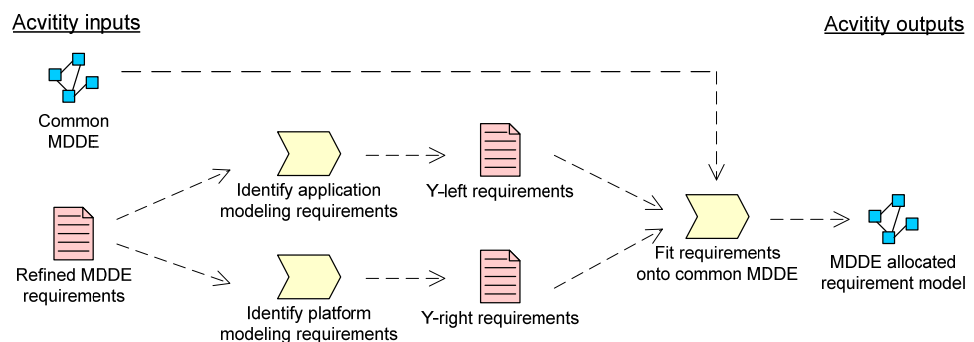


Figure 12. Requirement Allocation activity diagram.

The MDDE design phase contains four work definitions:

- Y-chart refinement (Figure 13). This work definition takes the MDDE allocated requirement model, created in the requirements phase, as input and produces a refined Y-chart structure as output. The first two activities identify patterns in the requirements of each Y-branch (application and platform). These patterns are used to create a more detailed Y-chart structure based on the identified patterns.

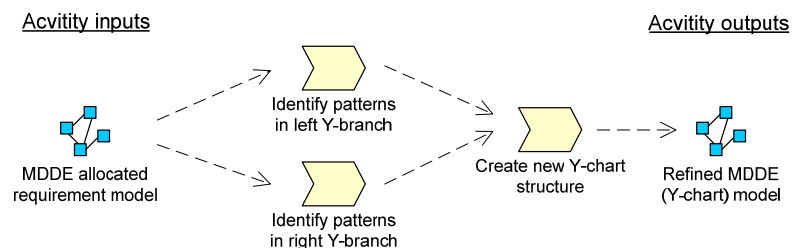


Figure 13. Y-chart refinement activity diagram.

- Define abstraction level concepts. The goal of this work definition is to identify the main concepts that will be needed to model the application/platform and pattern models at each branch of the Y-chart.
- Y-chart domains & traceability. This part of the work focuses on grouping the concepts that were defined in the previous work definition in order to get a first idea of the needed metamodels. Furthermore, the requirements for traceability are identified here and required link types can be established between the domains.

MARTES	Specification of the Model Driven Engineering Process	Page : 24 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

- MDDE method content refinement. This last work definition is needed to adapt the methods to the refined Y-chart.

The last two phases will not be discussed in detail here. They are about the formalization of transformations (allocation steps) and metamodels, followed by their implementation. We address transformations in a later stage because they can only be usefully described if stable metamodels are available. The implementation phase can be short in practice because many available tools support automatic generation of metamodel implementations. Therefore, the manual implementation can be limited to a number of modifications and additions. Most of the work in this phase will probably go to the implementation of transformations.

3.3.3 Conclusions

We have defined a first step towards a MARTES process framework that contains two main elements (see also Figure 9)

1. A **common MDDE** that serves as a shared starting point for developing a partner-specific MDDE. The common MDDE is composed of two parts:
 - a. The initial **Y-chart infrastructure content** with application, platform, allocation, allocated and implementation models, preferring the UML and a RTES profile and EMF/ATL as implementation technologies.
 - b. Accompanying, initial, **Y-chart method content** to guide the use of the Y-chart infrastructure content.
2. The **MDDE refinement process** – a specialized MDDE development case – that assists the MARTES partners with the specialization of the common MDDE (Y-chart infrastructure method content).

We found the practice of explicitly modeling processes very helpful. It forced us to really think about the activities that need to be carried out in the context of the, not yet so mature, Model Driven Development. Especially in such cases, it is important to constantly update the running process (e.g. after each iteration) and keep the process well-documented at all times. In the case of developing an MDDE for MARTES this practice applies to (1) the application process that needs to gradually incorporate the MDDE method content and (2) the MDDE process itself, which we refined after establishing an initial MDDE structure.

We have used SPEM 1.1 as the process modeling language, which provided some support but did not prove so easy to use, especially due the lack of tool support. The upcoming SPEM 2.0 standard appears to address many issues and preliminary tool support is already available in the form of the Eclipse Process Framework. Therefore we will use the latter one from now on.

3.4 Example of MARTES Process descriptions

The common conceptual modeling approach of MARTES is based on the Y-chart model as explained in Section 3.3.1. According to the generic MDDE development process, this approach can be applied in several ways in order to adapt to the different product and technology domains of MARTES partners.

In order to achieve enough flexibility to be applicable to any software engineering process, the proposals for SPEM2.0 (see Section 5.2) pursue the separation of method contents from

MARTES	Specification of the Model Driven Engineering Process	Page : 25 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

their application processes. The methods will be described independently and self-sufficiently in libraries categorized according to disciplines.

The generic disciplines identified in MARTES are as follows [v]:

- Requirement and Specification Modeling
- Application Modeling
- Execution Platform Modeling
- Allocation Modeling
- Implementation Oriented Modeling

Each discipline then contains a number of tasks providing step-by-step descriptions of how the development goals are achieved. These are considered as method contents in the SPEM2.0 proposal.

3.4.1 MARTES Requirement and Specification Modeling

3.4.1.1 Method Contents Description

As there are several proposals for SPEM2.0 that can still evolve remarkably, the following method contents descriptions related to MARTES Requirement and Specification Modeling are informal, apply loosely SPEM2.0 proposal contents, and use SPEM1 notation. Method content packages contain method elements like role definition, work definition, artifact definition and guidance definition.

Figure 14 outlines the contents of DefineUsageScenarios package.

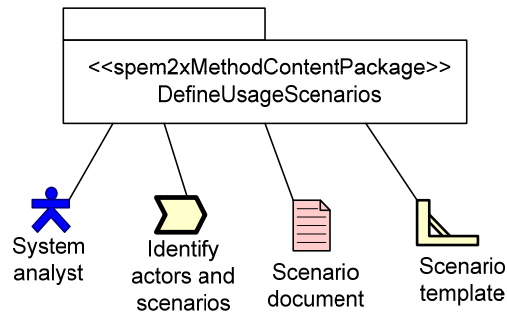


Figure 14. MethodContentPackage DefineUsageScenarios

Figure 15 outlines the contents of ModelUseCases package.

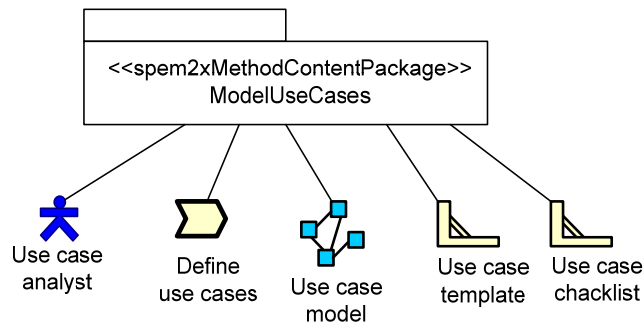


Figure 15. MethodContentPackage ModelUseCases

MARTES	Specification of the Model Driven Engineering Process	Page : 26 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

Figure 16 outlines the contents of RefineUseCases package.

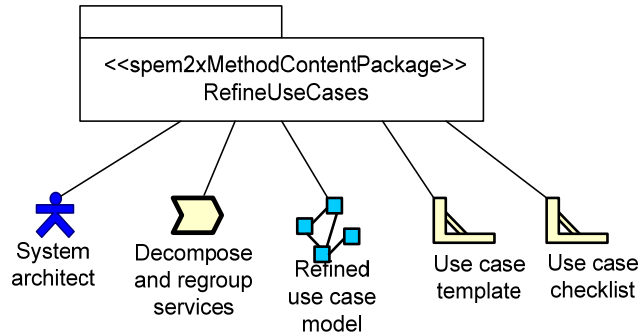


Figure 16. MethodContentPackage RefineUseCases

Figure 17 outlines the contents of ValidateUseCases package.

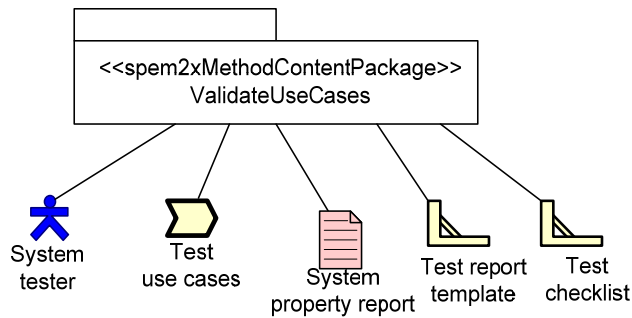


Figure 17. MethodContentPackage ValidateUseCases

Figure 18 outlines the contents of CreateSpecifications package.

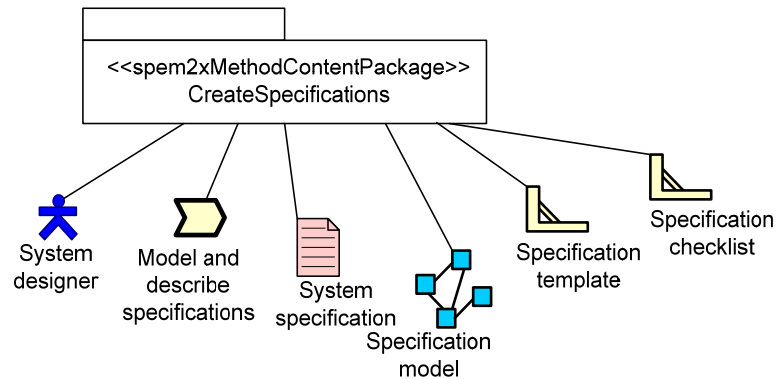


Figure 18. MethodContentPackage CreateSpecifications

3.4.1.2 Process Modeling Description

Application of method contents in a process in the form of an activity diagram is depicted in Figure 19.

MARTES	Specification of the Model Driven Engineering Process	Page : 27 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

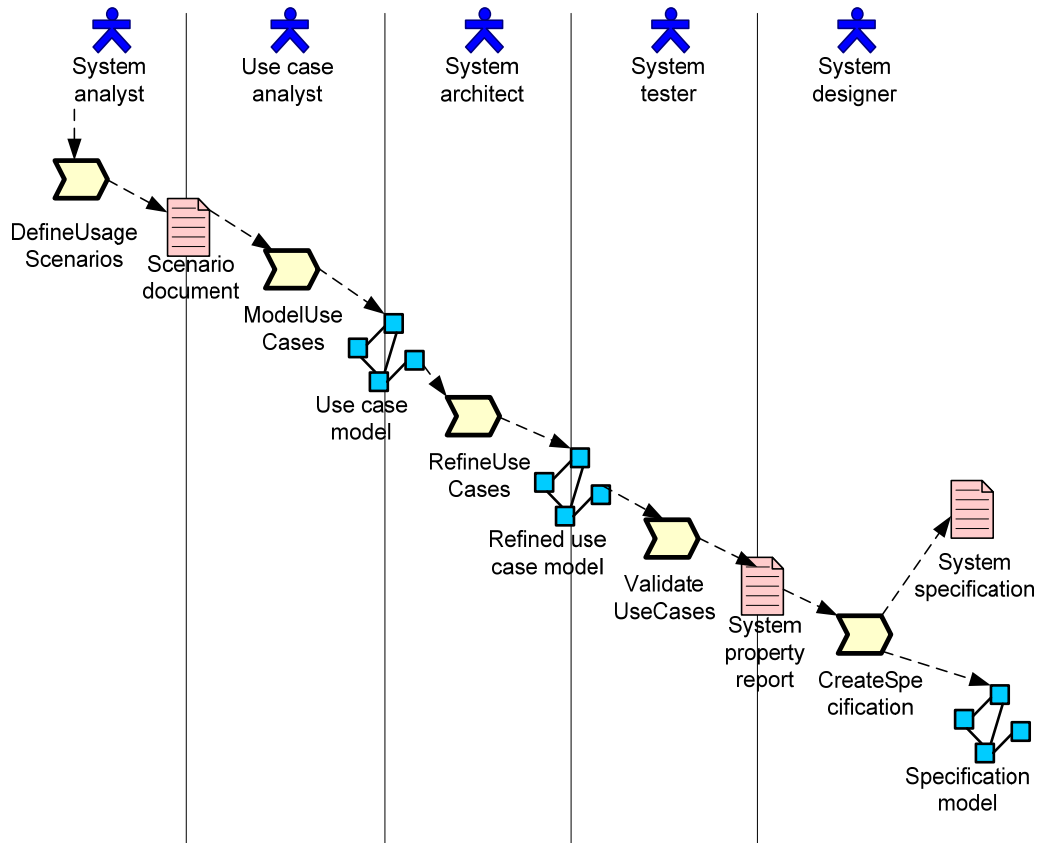


Figure 19. Requirements and Specification Modeling with Use Cases.

3.4.2 Process for translating from UML to SystemC

During the development process there is a need for translation from UML to SystemC. This can be done in a number of occasions during the development process. The translation is carried out as described in the following translation process.

The translation process consists of three major steps:

- Preparation for translation
- Translation
- Usage of translation results

The steps can be describes as in Figure 20.

MARTES	Specification of the Model Driven Engineering Process	Page : 28 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

Step	Input	Output	Activity
1. Preparation for translation	UML model	UML model adhering to the MARTES SystemC profile	Adjust the input model in order to make it conform with the profile
2. Translation	UML model adhering to the MARTES SystemC profile	SystemC representation	Tool supported translation
3. Usage of translation results	SystemC representation	Simulation results	Simulation of functionality, analysis of performance

Figure 20. Summary of UML to SystemC translation process.

During step 1 the input model is transformed to conform to the chosen profile. The input model can for the first version of the translation tool be at the Programmer View (PV) level and represent the application model. The first version of the tool supports the application model at the PV level and later version are intended to support also an Allocated model.

During step 2 the UML model is converted to a SystemC representation using the translator implemented as an add-in of the UML modeler tool as Telelogic Tau G2 for example.

During step 3 the SystemC representation is used to verify the functionality through simulation. It can also be used as input to performance evaluations.

The translation process can be used iteratively during the development process. First an initial UML model is translated analyzed, which results in changes and an updated version that can be analyzed n the same way.

3.4.3 Process for translating from UML to Java

Since it is necessary the translation from UML to Java in development process, it is necessary to describe how to carry out this translation. It takes several steps, those steps are equal to the previous section described ones, that means:

- Preparation for translation: A UML model is used for adding its information to the MARTES Java Profile; it guarantees UML model is according with the MARTES profile.
- Translation: The model obtained in the previous step is used to compose Java Classes. In order to accomplish this goal several tools are available for instead plugins for Eclipse. We can use some open standards like XMI (XML model interchange) to facilitate this translation.
- Usage of translation results: once Java Classes are generated is necessary to test in order to check functionality and behaviour on the target hosts.

As it is said in the previous section, this whole process can be used iteratively during the development process for taking into advantage of the feedback provided by the process.

MARTES	Specification of the Model Driven Engineering Process	Page : 29 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

4 MDE Traceability

The goal of traceability is being able to establish relationships between the different work products in a software development process. We first explain in Section 4.1 how we define traceability and then classify different traceability approaches along two axes in Section 4.2. We continue in Section 4.3 by discussing two possibilities to represent traceability information and discuss in Section 4.4 on how to make that information interchangeable between different partners. We make a selection in Section 4.5 of what we think is the best combination of techniques to use for MARTES. Finally in Section 4.6, we present an approach allowing to use traceability information as input for further model transformations.

4.1 What is Traceability?

Traceability is often associated with the tracking of requirements from less formal (documentation, requirements) to more formal (models and software components) artifacts throughout the software development process [vi,vii]. However, in the case of transformations it also refers to the logging of transformation operations and their source/target model element mappings without having anything to do with requirements directly [viii,ix]. In other cases it might even signify the link between an artefact and the person responsible for that artefact or the time it was created, etc. One of the problems with traceability is that it simply has no commonly accepted definition.

In the context of MDD we define traceability in a very broad sense as *the ability to extract historical information out of a current model*. This definition also comprises traceability throughout manual or automatic model transformations, since models also evolve over time by applying consecutive transformations, for example by model refinement. Put in other words, the definition states that traceability is the ability to see what happened to the model in the past to get to a model in the present or it describes the relationship between an earlier model (usually more abstract) and a later model (more concrete).

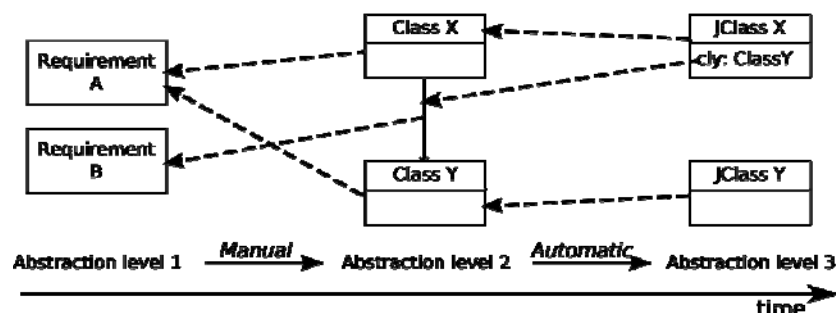


Figure 21. Traceability links, shown as dotted arrows

Figure 21 illustrates our definition with an example. In this example we have three models ordered from higher to lower abstraction levels, which is in many cases equivalent to the ordering according to the creation time (hence the timeline). The first level contains informal requirement documents, which are manually converted to a class diagram. The dotted arrows are traces that indicate that class X and Y are introduced to fulfil requirement A and

MARTES	Specification of the Model Driven Engineering Process	Page : 30 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

the association between X and Y to fulfil requirement B. The next transformation could be executed automatically and the traces are now between model elements of two different abstraction levels. In the next section we apply a coarse classification to traceability approaches.

4.2 Classifying Traceability

Since the definition that we gave in the previous section is very broad, there is room for many different types of traceability. In this section we classify traceability across two orthogonal axes.

4.2.1 Requirements vs. Transformation Traceability

Requirements traceability is used particularly in software evolution management. When a stakeholder issues a new or changed requirement we want to be able to easily localize the (modeling) artifacts that are influenced by that requirement. Managing the links between (often textual) requirements and implementing artifacts at several levels throughout the development process is generally referred to as requirements traceability. A more specific flavor of the general requirements traceability, the refinement/abstraction relationship, is discussed in [vii]. A reference metamodel for representing requirements traceability links within the UML is described in [vi]. The authors of that paper choose to integrate their mechanisms into the UML in order to represent all software artifacts in one model. In a sense transformation traceability can also be seen as a specific type of requirements traceability and can be integrated with and augment other approaches.

4.2.2 Generic/Full vs. Specific Traceability

We distinguish between generic/full traceability and specific traceability. Full traceability means that every possible model element can be traced back to another model element or elements that triggered its creation. It can be automatically realized by a transformation engine in the case of automatic transformations; every changed element in the output model is then linked to its counterpart in the source model, usually also linked to the responsible transformation rule (e.g. QVT and ATL [ix]).

One of the potential problems with this kind of automatically generated traceability is its dependence on the transformation implementation, meaning that the manner in which a transformation is implemented determines the form of the traces. An example of a problem is illustrated by the transformations given in Figure 22 and Figure 23. Both these transformations, written in pseudo-code, accomplish the same result but the first one uses two rules and the second does it with only one rule. This means that in the first case, two traces will be created: one labeled with the first rule and one labeled with the second. In the second case there are two possibilities: create two traces with the same label or create one many-to-one trace. Either way, the traces generated by the alternative implementations will at least differ in the trace labels. A consequence is that subsequent transformations need to have some knowledge on the implementation details of previous transformations if it wants to make use of these traces. In other words, generic/full traceability only provides semantically weak traces unless we have internal implementation knowledge. Another disadvantage of

MARTES	Specification of the Model Driven Engineering Process	Page : 31 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

this type of traceability is that is rapidly overcrowds a model with traces. This makes traceability complete but not necessary very usable.

```

Attribute2Attribute {
    source: attribute
    target: attribute
}
Attribute2Accessor {
    source: attribute
    target: operation
}

```

Figure 22. Copy attributes and add an accessor – variation 1.

```

Attribute2Attribute&Accessor {
    source: attribute
    target: attribute, operation
}

```

Figure 23. Copy attributes and add an accessor – variation 2.

Some of the disadvantages of full traceability can be solved by using specific traceability. Specific traceability does not aim to link every source/target tuple with a trace but rather aims to create traces that have a more semantically rich meaning without overcrowding the model with traces. To get semantically rich traces it is necessary to first define the types of traces that may be used. By doing this in a separate step, the labeling of the traces becomes independent of the implementation and as consequence traces become more reusable.

The biggest disadvantage of the specific traceability approach is that the generation of traces will have to be programmed manually into the transformation rules. Figure 24 gives an example where a trace is introduced as an extra target within a transformation rule; notice that also the specific type of the trace is indicated.

```

Attribute2Attribute&Accessor {
    source: attribute
    target: attribute, operation
    target': trace(AccessorType, attribute, operation)
}

```

Figure 24. Explicitly creating traces.

It is also not very clear how to determine which kinds of specific links could be useful for subsequent transformations. In Section 4.4 we discuss the problem of choosing the types of trace links further. In the next section we discuss two alternatives of representing traces.

4.3 Traceability Representation

Independently of the type of traceability we choose to use, we also need to choose a way in which to represent the traces. Also in this area there are some alternatives. We can choose

MARTES	Specification of the Model Driven Engineering Process	Page : 32 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

to embed the traceability in the models themselves (intra-model) or store the information in a separate external model (extra-model).

4.3.1 Intra-Model Traceability

The intra-model approach requires us to explicitly extend ours metamodels with traceability concepts. Doing so introduces certain pollution into the models. There is also no straightforward solution to trace elements across different models, certainly if we do not want to intertwine all models. An advantage of the intra-model approach is that all traceability information is grouped in one place, together with the other model information. This makes it easier to access traceability information since we only need to know about one metamodel. A simple way of implementing intra-model traceability in UML is by using a UML profile. This profile can specialize the existing *dependency* element that enables the insertion of traces within model without even having to extend the UML metamodel.

4.3.2 Extra-Model Traceability

The extra-model approach uses a traceability model that is external to the application models. A traceability model conforms to a dedicated traceability metamodel that exclusively describes traceability concepts. A trace links is then established by instantiating a trace element in the traceability model that refers to the appropriate elements in other models. As a consequence, we need a mechanism that enables us to refer to several heterogeneous model elements from within our traceability model.

Besides the fact that the extra-model approach is the best conceptual solution, it has some practical disadvantages. One of the problems is how to keep different related models synchronized. Every possible operation of a model would have to be aware of the external traceability information because a change to one of the traced models could break the traceability model. Using a separate traceability model also makes transformations more complicated since they need to take an extra input if they want to use traceability information. The most part of the issues concerning extra-model traceability can however be solved with the right tool support, by making the traceability model transparently accessible.

4.3.3 Traceability Metamodel

A traceability metamodel captures how model elements may be related by trace relations and defines the semantics of such relations. It is often necessary to distinguish between different kinds of traces. For example, it is interesting to know whether a trace denotes a relationship between a textual requirement and an architectural element or indicates a refinement within the design. The required kinds of traces are often highly project dependent [x]. In this subsection, we make a concise comparison between two different traceability metamodels that both facilitate specific kinds of traces.

MARTES	Specification of the Model Driven Engineering Process	Page : 33 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

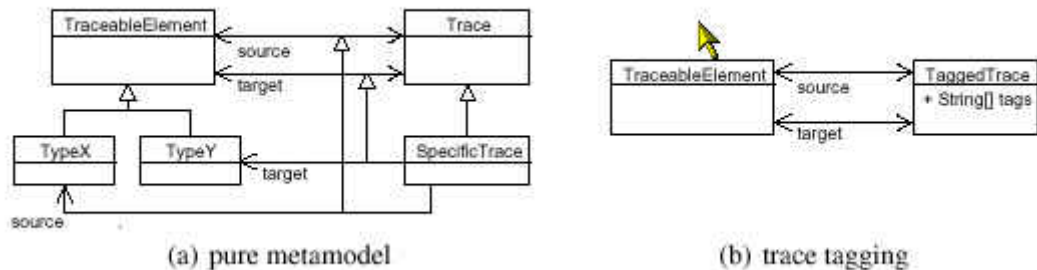


Figure 25. Two extreme traceability metamodels.

In a pure metamodel approach, the traceability metamodel specifies every kind of trace we may need. Figure 25a shows a snapshot of such a metamodel. The two top classes indicate that a Trace can relate two TraceableElements. If we need a more specific kind of trace it suffices to subclass Trace and specialize its associations (see SpecificTrace in the figure). The most notable advantage of this approach is that we can precisely specify both usage constraints and semantics of each trace kind at the metamodel level. The main drawback is its lack of flexibility: every change in traceability requirements needs to be reflected (hard-coded) in the metamodel. Project-specific trace kinds are expressed directly in the metamodel, which limits potential reuse to one project.

The trace tagging approach uses a simple and general traceability metamodel (see Figure 25b) and allows users to annotate generic traces with attributes or tags. A similar approach is used in specialized tools such as Telelogic DOORS. The kinds of traces are represented by tags and are defined at the model rather than at the metamodel level. This prevents us from specifying precise semantics and usage rules in the metamodel. A user can add any tag without having to adhere to any rules. At the same time this kind of flexibility is its biggest advantage. The metamodel never needs to be changed, hence this kind of metamodel can be reused in any project.

We will show later that, because of its generality and flexibility, the trace tagging approach is best suited for integrating traceability into transformation compositions.

4.3.4 Automatic Trace Generation

Automatic model transformations can generate traceability information along with the target model(s). According to [xi], transformation approaches either have dedicated support for traceability or rely on the developer to encode traceability as a regular output model. In any case, it is favorable to incorporate traceability generation into the transformation as opposed to producing it manually, no matter whether we apply the pure metamodel or tagging approach.

The major advantage of dedicated traceability support is that we get the traceability model(s) at an extremely low cost; the developer has to do little to no additional effort. A practical disadvantage is that the traceability metamodel is then fixed and may not be standardized among different transformation engines. Also, the level of granularity of the traces may not be the same.

MARTES	Specification of the Model Driven Engineering Process	Page : 34 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

Alternatively we can treat traceability as a regular output model of the transformation and incorporate additional transformation rules to generate it. The choice of metamodel is then completely at the discretion of the developer and does not depend on the transformation engine. The drawback is that additional effort is required to add traceability-specific transformation rules, which also pollute the implementation. An approach that partly solves these issues by automatically generating the trace-related transformation rules was proposed in [xii].

Since our research is focussed on composing many subtransformations and combining different transformation technologies, we can only assume a generic traceability metamodel (i.e. Figure 25b) as the least common denominator of all transformation technologies. Furthermore we assume that each transformation creates a separate trace for each generated target model element; this link points to the source element(s) from which the target was created. Hence, no model element exists that is not fully traceable back to its source throughout the transformation chain.

4.4 Traceability Interchange

Related to the representation of traceability discussed in the previous section is the issue of standardization of traceability models. It is important to think about what information exactly will be stored in the traceability model. Just having generic links between model elements usually is not enough, for example, you might want to distinguish traces to textual requirements, between elements of different types of models, traces to physical entities such as the responsible developer, the last changed date, etc. The type of information that needs to be kept in a traceability model probably differs from project to project.

We could therefore ask ourselves if it would be useful to have a single common traceability metamodel ? The advantage of having one is that every operation on a model can always understand the traceability information. The disadvantage probably would be its generality, preventing the inclusion of additional and project specific information. A possible solution is to agree on a basic but extensible core traceability metamodel that can be adapted to specific needs while still allowing interoperability and interchange between tools and people. This would also mean that an agreement is reached about the representation of the traceability information.

4.5 Choosing the Right Technique

For each project one must choose the right combination of the discussed possibilities concerning traceability. The MARTES project is characterized by a large number of partners that want to establish a shared methodology for model-based development. As a consequence the different requirements of the different partners influence our choices.

It seems logical to choose for a common core traceability metamodel that embodies most of the common requirements. We do not want to include all the requirements of each partner in order not to bloat the metamodel. Each partner can then extend this core model with its own additional requirements, as explained in Section 4.4. We argued for a combination of specific and full/generic traceability (see Section 4.2). The common part would be specific, addressing only that kind of traceability information that adds additional information to the

MARTES	Specification of the Model Driven Engineering Process	Page : 35 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

model that can be reused by the different partners and cannot be derived otherwise. The full/generic part is left to the partners and is not exchanged.

The best way to represent the traceability information in the MARTES case is by using the extra-model approach since this can make traceability across different models, currently used by different partners, easier.

A final decision that must be made, which not only applies to traceability, is the technology that will be used to preserve the information. The most convenient way is to select the same technology that is chosen for the other MARTES models.

4.6 Traceability as Input for Model Transformations

Some model transformations require more information than can be derived from its source model(s) in order to generate a meaningful target model. For example, a transformation with two source models needs to know how their respective model elements relate; these relations often only exist implicitly as part of the transformations developer's knowledge. In this section we show that traceability models, who can be automatically generated as part of any model transformation, contain explicit inter- and intra-model relations that are valuable to subsequent transformations. We explain how to extract this information and propose a number of additions to current transformation techniques that are needed to completely open up traceability information to transformation developers.

4.6.1 Introduction

Classically, the creation and maintenance of traceability information has been a manual and labor intensive task. The rise of Model Driven Development (MDD) eases this problem by introducing model transformations, which can generate basic traces automatically. MDD recognizes the need of many different intermediate models to represent a system, which results in an abundance of readily available traces that interrelate every piece of the system. Typical uses of this traceability information are: showing that requirements are met, analyzing impact of changes, propagating changes, etc. In this section we present a new use of traceability information in the context of transformation chains. A transformation chain or transformation composition is a network of many subtransformations, each contributing a small part to a larger transformation goal. Each subtransformation also produces traces that contribute to a global traceability graph. This graph interrelates disparate models that represent different aspects of a system (e.g. functional, security and persistence model), produced as part of a transformation chain. In some cases we need to know exactly how these aspects are related in order to perform further transformations (e.g. which database table stores a particular attribute from the functional model). This kind of information usually cannot be extracted from the individual models but is hidden in the traceability models. We will show how generated traces can provide non-trivial inter- and intra-model relations without imposing many additional requirements on the models or transformations. We define trace navigation as an operation to derive the required information and propose trace tagging to enrich the information contained in the traceability graph. Finally we discuss how these concepts can be integrated in current transformation technologies.

MARTES	Specification of the Model Driven Engineering Process	Page : 36 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

4.6.2 Leveraging Traceability Models

In an MDD approach we use many models that offer different views on a system: different levels of abstraction, different sub-aspects of a system, etc. Many of these models are derived through automatic transformations and are hence highly interlinked through automatically generated traceability models (as discussed in 2.2). In 3.1 we show that the information contained in traceability models can be leveraged for development of further transformations. We discuss an elaborate example in subsection 3.2.

4.6.2.1 Discovering Useful Trace Information

Figure 26a and Figure 26b present examples of transformations and models along with their generated traceability models. Each curved line in the figure represents a single trace and is part of a traceability model. For example in Figure 26a there are in fact three models: UML-1, UML-2 and the traceability model that connects these two. (a) intra-model (b) inter-model.

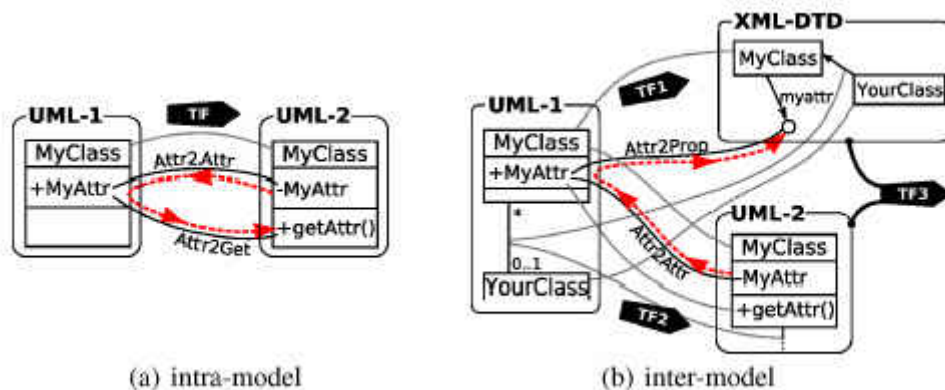


Figure 26. Deriving model relations from past traceability information.

Figure 26a shows a transformation TF from source UML-1 to target UML-2 that adds accessor methods (i.e. getters) to classes and makes corresponding attributes private. We focus on traces Attr2Attr and Attr2Get, labeled according to their transformation rule. The dotted arrows on the figure indicate that we can navigate back and forth through the traceability model and the UML-1 model to discover an intra-model relation between MyAttr and getAttr() in the UML-2 model. A subsequent transformation can use this information, which matches attributes to their accessors, for example to generate a detailed implementation for accessors. As a result, the concern of naming the accessor and making the attribute private (transformation TF) can be separated from the implementation generation. The necessary information is implicitly passed via the generated traces. Without traceability, a subsequent transformation would only be able to guess the relation between accessor and attribute (e.g. by matching strings).

A second example is shown in Figure 26b. Class model UML-1 is transformed into an XML data type definition model (XML-DTD) [xiii] and a refined class model UML-2. This transformation is accomplished in two parallel steps (TF1 and TF2); TF2 is equivalent to the transformation in Figure 26a. Suppose that we now introduce an additional subsequent

MARTES	Specification of the Model Driven Engineering Process	Page : 37 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

transformation (TF3) to generate a persistence layer that can load objects from an XML file. This transformation needs to know, amongst others, the exact relations between class attributes and XML properties in order to produce a valid target model. If we solely use UML-2 and XML-DTD as inputs it is hard to find these relations. Resorting to the traceability models offers a solution. For example, the XML property that corresponds to class attribute MyAttr can be found by navigating back (Attr2Attr) and forth (Attr2Prop) through the traceability model (see dotted arrows). This example shows that we can also derive inter-model relations from traceability models.

In the former paragraphs we have used 'navigate' whenever we combined the information from one or more traces to derive relations between transformation input model elements. We consider the union of all traceability models and regular models as a graph G where elements from the regular models are vertices V and traces are edges E. We can hence give a definition for trace navigation.

Definition 1. Given an input vertex v_i and a set of navigation target models M_t , a trace navigation is any operation that takes v_i as input and yields a collection of vertices V_o as output. All the elements of V_o are reachable from v_i and are owned by a model in M_t .

4.6.2.2 Example: Persistence with a Relational Database

In this subsection we present a realistic example where a relational persistence layer is generated by a composition of a transformations. The complete composition is shown in Figure 27.

In the upper part, TFA1 and TFA2 subsequently transform the initial class model into an entity-relationship model (ER) [xiv] and corresponding relational database tables (RDB/SQL). Notice that the transformation from entity-relationship to database tables (TFA2) is not a trivial one-to-one mapping; only two tables are introduced to represent three elements of the ER model. In the lower part, TFB1 and TFB2 transform the class model into a simplified class model UML-2 (taking away the association class) and subsequently a JAVA model. The figure also shows all the traces that were created by each subtransformation.

Having this many intermediate models has the advantage that different concerns are treated separately in different transformations and are visible in separate, highly specialized models. This narrows the scope of each subtransformation, making them easier to implement, and offers a dedicated view to each domain expert – i.e. a database specialist considers the ER model while a Java programmer only looks at the JAVA model.

After executing all the transformations, we end up with the current models: RDB/SQL and JAVA. At this point we introduce a new transformation TFNEW (see Figure 27) that adds database access code to the JAVA model; part of the resulting model is shown in the rightmost part of the figure. Transformation TFNEW takes both RDB/SQL and JAVA models as input since a persistence layer involves getting values from the tables in the database (RDB/SQL model) and storing them as class attribute values of the JAVA model.

In Figure 27 we show how we can relate classes to database tables in order to generate the correct sql query for the Person class. Just by navigating the traces through the old models,

MARTES ITEA 04006	Specification of the Model Driven Engineering Process Deliverable ID: 1.7	Page : 38 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final Confid : Public

we end up with the Employee table (follow the lines marked with dots), which is indeed the table where the information about Person is stored; note that we also get the corresponding primary key. It would be very difficult to find this relation without the traceability information since there is no one-to-one correspondence from the RDB/SQL model structure to the JAVA class structure. It is easy to see that we can also use the traces to find corresponding table fields for the attributes of the Company and Job class, for example a Job's salary attribute corresponds to j salary column in the database. Because of space restrictions we leave it up to the reader to find other relations hidden in the traceability information. Two other examples are given at the bottom rightmost corner of Figure 27.

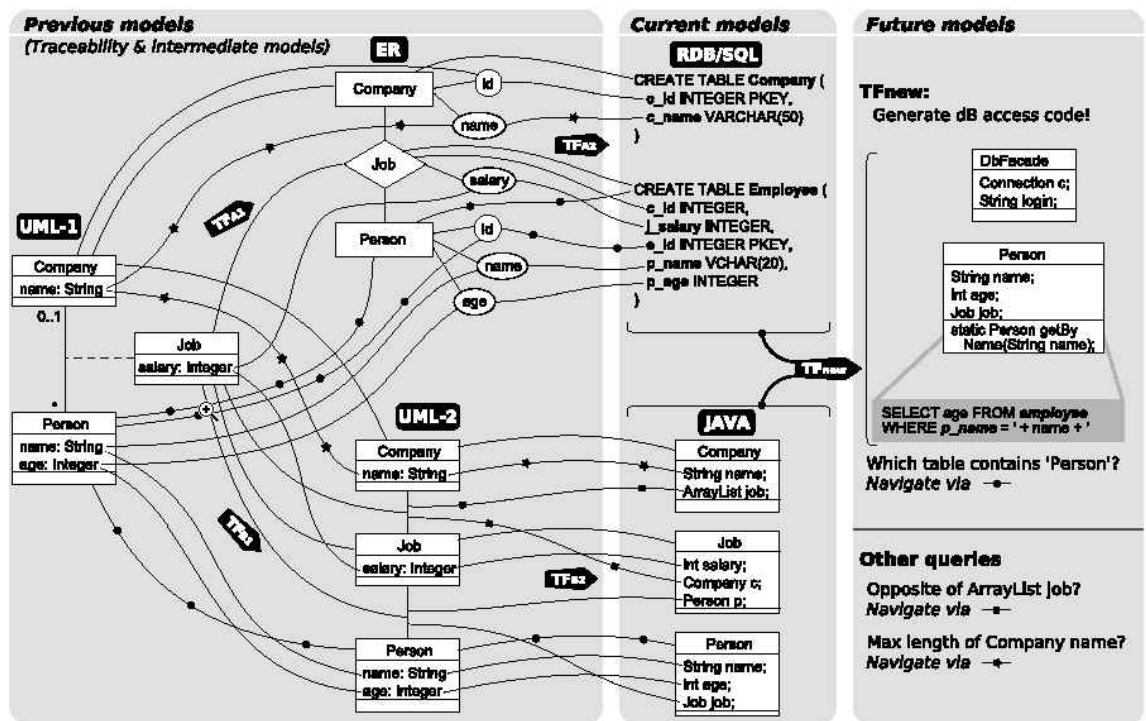


Figure 27. Using traceability in a transformation chain to generate a relational persistence layer.

In this and the previous subsection we have shown how we can extract valuable information from a global traceability graph that cannot be extracted from models individually but is sometimes required to perform meaningful transformations. A key observation that can be made at this point is that, although we heavily use the traceability models to derive relations, a transformation only needs to consider its actual inputs. The navigation through the traceability models can be handled transparently by a generic 'trace query' operation. As a consequence, a transformation does not have to depend explicitly on each one of the previous (traceability) models (see also Section 5).

MARTES	Specification of the Model Driven Engineering Process	Page : 39 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

4.6.3 Producing Unambiguous Traceability Models

As allowed by Definition 1, a trace navigation can return more than one result. In many cases the required element can be selected by filtering on the element's type (see also Section 3.2). If the result of a navigation yields several elements of the same type however, the obtained information will be ambiguous. We retake the example where an accessor method is generated for each class attribute and we now include generation of mutator methods (i.e. a setters) in a separate transformation (see Figure 28). If we start navigation from attribute MyAttr, we get a collection of two operations as result: both mutator and accessor (see dotted arrows). Since they are of the same type (Operation), it is impossible to filter out the desired result.

To solve this issue we need to add additional semantics to the traces that allow to distinguish different trace paths. In case of the example we need to be able to find out which trace path leads to the accessor and which leads to the mutator. Since we use a generic traceability metamodel, an appropriate way to add these semantics is by applying the trace tagging approach (explained in 2.1). This way we do not limit the possible semantics of the traces by the traceability metamodel; these semantics are very dependent on the type of transformation and it seems quite impossible to capture all possibilities a priori as in a pure metamodel approach (see 2.1).



Figure 28. Navigating tagged trace links.

In Figure 28 we apply two consecutive transformations: first we add mutators (TARGET1) and subsequently we add accessors (TARGET2) and tag the traces appropriately with 'set' and 'get'. Further transformations can now unambiguously find both mutators and accessors for each attribute in model TARGET2 by navigating via the appropriate traces. As long as a 'set' trace is encountered on the navigation path we get the mutator; when a 'get' tag is encountered, we get the accessor. We refer to a navigation that takes tagged traces into account as a qualified navigation. The following definition uses the same assumptions as Definition 1.

Definition 2. A qualified trace navigation for tag t is a navigation where each path from input vertex v_i to any of the result vertices in V_o contain at least one edge (trace) tagged with t .

In case the tagged trace models, produced by previous transformations, still yield ambiguous trace navigation results it is up to the transformation assembler (who specifies a transformation composition) to decorate the traces with additional tags.

MARTES	Specification of the Model Driven Engineering Process	Page : 40 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

4.6.4 Integrating Traceability in Transformations

We have shown that automatic model transformations can generate traceability models at a very low cost and that we can navigate through traces to extract useful information. In this section we identify three transformation areas that need to be extended in order to fully implement our approach. We also propose early ideas to extend these areas.

Production of Traces

It is not very hard to produce traceability models as part of model transformations. Some transformation languages even offer dedicated support for this purpose (for an overview, see [xi]). Nevertheless, we believe that it would be useful to see how this can be improved, keeping the trace tagging metamodel in mind. Adding dedicated trace tagging syntax to transformation languages could be very useful, for example allowing annotation of each transformation target element with a tag that is transferred to a corresponding trace at execution time (see Listing 1.1).

MARTES	Specification of the Model Driven Engineering Process	Page : 41 of 52
		Version: 1.0 Date: 14/06/2007
ITEA 04006	Deliverable ID: 1.7	Status: Final Confid : Public

```
rule Attr2Set
source (AttributeIn)
target (OperationOut) tag 'set'
```

Listing 1.1. Example syntax for trace tagging (in pseudo code).

Traceability Queries

In order for transformations to access traceability information (through the global traceability graph), they will need to include both traceability models (edges) and past regular models (vertices) as additional inputs. This has very negative effects on various aspects of the transformation. First of all reusability goes down since the transformation is directly dependent on all the intermediate results produced by previous transformations. Secondly, the specification of the transformation becomes cluttered with many additional models that are not used directly in the transformation but are solely used to derive relations between the real input models. Finally, trace navigation needs to be encoded in the transformation itself, cluttering its implementation.

However, as briefly mentioned in Section 3.2, a transformation does not have to depend directly on all the previous models if we factor out the (qualified) trace navigation operation. We propose to introduce the `traceVia(tag: Set(String)): Set(oclAny)` operation in OCL as the single access point to the traceability information. In Listing 1.2 we use this operation to find out if an operation is a mutator. This approach would eliminate most of the disadvantages described in the previous paragraph.

```
rule RenameMutator s
source (OperationIn)
when in.traceVia('set')->select(oclIsTypeOf(Property))->
notEmpty()
target (OperationOut)
out.name = 'my' + in.name
```

Listing 1.2. Example syntax for trace navigation (in pseudo code).

Declaration of Traceability Usage

Each transformation needs to have a clear specification so that it can easily be reused and composed in a transformation chain. In the first place, this comes down to a specification of in and output models. But, if transformations also depend on traceability information from previous transformations, the specification also has to include that information. We think that this can be accomplished by specifying required and provided trace tags. Each transformation can define its own tags independently of other transformations, improving reusability.

MARTES	Specification of the Model Driven Engineering Process	Page : 42 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

Whenever the transformation assembler connects transformations it might then be necessary to do a semantic mapping between required and provided tags. More research is needed to find a suitable set of mapping strategies.

4.6.5 Conclusions

Currently, most model transformations do not take the results of previous transformations into account. These intermediate results are a collection of regular and traceability models; the latter can be produced at very low cost in an MDD setting.

In this section, we considered intermediate models as part of a global traceability graph and showed that this graph can be used to derive useful relations between (inter) an within (intra) input models of subsequent transformations. These relations cannot be derived from individual input models but are sometimes required to specify meaningful transformations. We defined the notion of (qualified) trace navigation as a metamodel independent way to search for such relations. We introduced semantic annotations (tags) on the traceability models to avoid ambiguities during trace navigation.

Finally, we discussed three areas in transformation techniques that need to be extended in order to enable the use of traceability in transformation development. These are integration of trace creation (1) and trace navigation/querying (2) in transformation languages and (3) declaring traceability usage in a transformation specification. We are currently extending ATL [xii] with the traceVia operation that we proposed as a solution for (2). We will use this extended version of ATL to further validate our approach.

MARTES	Specification of the Model Driven Engineering Process	Page : 43 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

5 MDE and Standardization

This chapter presents some standards that are of influence on the definition of the MARTES MDD Process Framework, namely the DO-178B standard in Section 5.1, the standard for the development of airborne software in the civil aviation industry, and OMG's SPEM standard in Section 5.2, the Software Process Engineering Meta-Model that defines concepts for modeling software development processes.

5.1 DO-178B

DO-178B is the standard for the development of airborne software in the civil aviation industry. It is very process oriented and clearly defines the objectives, input and expected outputs of each process, without actually defining how to achieve these objectives.

5.1.1 DO-178B Processes

The DO-178B standard defines following processes and sub-processes:

- ***Software planning process***

The different plans define what activities will be performed in each process and how the output of these activities will satisfy the objectives of DO-178B.

- ***Software development process***

- *Software requirements process*

System requirements allocated to software are refined and further developed into high-level software requirements. Software high-level requirements describes the functionality of the software (i.e. *What* the software should do)

- *Software design process*

The software high-level requirements lead to a top-level architecture describing the components and their interactions. Based on the architecture the high-level requirements are further detailed into low-level requirements that describe more *how* the software satisfies the high-level requirements. Hence traceability needs to be established between the high-level and low-level requirements to ensure that no requirements are forgotten.

- *Software coding process*

From the low-level requirements, which are in fact a functional description of what a particular function or procedure should do, source code is developed.

Additionally, traceability needs to be established between the low-level requirements and the source code.

- *Integration process*

During the integration process, the source is compiled and linked into executable object code that is then downloaded into the target platform.

- ***Software integral process***

MARTES	Specification of the Model Driven Engineering Process	Page : 44 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

The integral processes are process that support and guide the development processes and are running in the background throughout the whole development process.

- Software verification process

This process consists of the actual testing, analyses and reviews. All these activities aim at detecting errors that might have been introduced during the different development process steps.

- Software quality assurance process

The software quality process has as objective to independently ensure that the processes as defined in the plans during the planning process, are followed and that the objectives are satisfied. The activities to achieve this are audits, milestones reviews and witnessing.

- Software configuration management process

All software items created during the software development (documents, source files, design files, test files, etc.) needs to be uniquely identified, base-lined and released. Also changes to these items, needs to be logged, evaluated, assessed and tracked till closure.

- Software certification liaison process

To obtain certification from the authorities, evidence and submittal data need to be delivered to the certification authorities to ensure that all objectives of DO-178B have been met.

5.1.2 MDD and DO-178B

As DO-178B is process driven and doesn't specify how to implement these processes, it allows the use different software development methodologies. Hence a MDE process can be used, although DO-178B will put some additional requirements on the tools used, if they reduce, eliminate or automate some activities.

As an example different kind of tools may be used during the different process steps:

- Requirements management tools to change, track and establish baselines of sets of requirements. These tools may also support requirements traceability.
- Software design tools that allow to define the software architecture and to link it to the high level requirements.
- Automatic code generators to automatically generate code from a model description.
- Configuration management tools, to take care of all configuration management issues for source files, documents, requirements, design files, etc.
- Verification tools to create test cases, test frameworks and report the achieved structural coverage from the testing
- Analysis tools like static code analyzers to find potential coding problems during the coding process.
- Rule checkers to check if the source code complies with the standard coding rules. Rule checkers may also work on the model level

MARTES	Specification of the Model Driven Engineering Process	Page : 45 of 52
		Version: 1.0 Date: 14/06/2007
ITEA 04006	Deliverable ID: 1.7	Status: Final Confid : Public

- Traceability tools that establish and manage traceability from the high-level requirements down to the source code and from the requirements to the test cases and test results

If in the MDD process tools are used during the development process to automate activities, like automatic code generators that transform a model into source code, DO-178B requires that this tool shall be qualified as development tool. This basically comes down to the fact that the part of the tool that performs the actual code generation should be developed according to DO-178B. The reason behind is that these tools may introduce errors directly into the software without being noticed.

Tools that automate certain verification activities and hence cannot introduce errors directly into the software, but may fail to detect software errors during testing, shall be qualified as verification tool. This means that their operational functions shall be specified and their operation shall be evaluated and assessed to check if the tool correctly detects and reports the potential software errors.

All other tools that cannot introduce errors nor fail to detect errors, doesn't need to be qualified.

Hence the DO-178B imposes additional requirements on the development of MDDE, in case the MDDE will automate processes and activities, by fully relying on the environment and tools without investigating the output manually.

5.2 SPEM

There is a need to choose a meta-model for the process modeling to be able to define a generic framework that can be specialized by different organizations and projects. One strong candidate is the on-going Software Process Engineering Meta-Model (SPEM) [xvi,xv]. SPEM is, as its name says, a metamodel that defines concepts for modeling software development processes. Modeling a process explicitly, with standard notations, forces us to think about process elements, the relationships between process elements and effects upon one another. In the context of MDD in general, it is considered good practice to codify best practices, in this case development processes, which are very often implicitly followed. One of the major contributions of SPEM is the provision of a common vocabulary to talk about processes. We give a short overview of the main elements of SPEM in the following paragraphs.

5.2.1 SPEM v1.1

The SPEM v1.1 metamodel [xv] is an extension of a subset of the UML v1.4, which means that the UML constructs in this subset can also be used in SPEM models (specially state and activity models). The standard also defines a UML profile, which enables us to express the complete metamodel by using only the standard UML syntax. A (semi-)dedicated syntax is available in the form of stereotype icons.

MARTES	Specification of the Model Driven Engineering Process	Page : 46 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

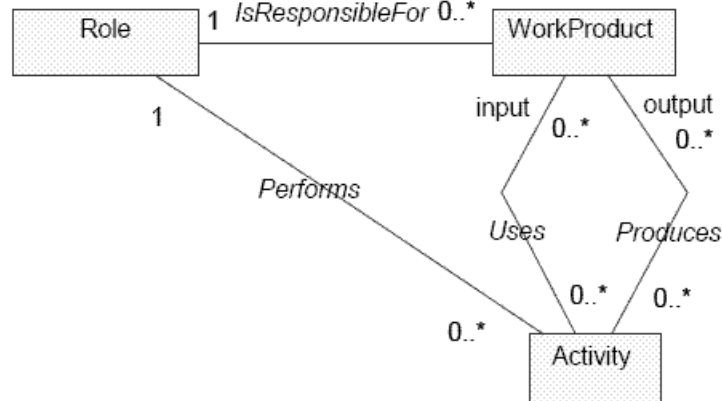


Figure 29. The conceptual SPEM model.

The core idea of SPEM is illustrated in Figure 29. A software development process is a collaboration between **process roles** (a collection of skills and responsibilities) that are involved in certain **activities** that produce or use artefacts called **work products**.

In order to give a quick overview of the most important elements in SPEM we use the following classification:

Work breakdown (or process structure)

A subset of the SPEM elements enable the structural decomposition of the work that needs to be done in a process:

- *ExternalDescription*: A textual description of any SPEM model element to serve as the user-visible surface of the process description.
- *Guidance/GuidanceKind*: Provides additional information to practitioners about the associated model element in the form of guidelines, techniques, metrics, examples, UML profiles, etc. (e.g. Java implementation guidelines that come with the UP).
- *Dependencies*: The two most important dependencies are *impacts* (indicates that modification from one WorkProduct could invalidate another) and *precedes* (between Activities or WorkDefinitions to indicate finish-start or finish-finish).
- *WorkDefinition*: A kind of operation that describes the work performed in the process. A WorkDefinition is related to the WorkProducts it uses through the *ActivityParameter* class, which specifies whether they are used as input or output. It is executed by one main *ProcessPerformer*. An *Activity* is the main subclass of WorkDefinition; it describes a piece of work performed by one ProcessRole: the tasks, operations, and actions that are performed by a role or with which the role may assist. An Activity may consist of atomic elements called *Steps*. In the UP for example, “Find use cases and actors” is an activity that is decomposed in small steps, from “find actors” to “check the results”.

Process Lifecycle

Describes how the process will run in terms of deadlines, milestones, etc.

MARTES	Specification of the Model Driven Engineering Process	Page : 47 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

- *Phase*: 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. In the UP: Inception -> Elaboration -> Construction -> Transition.
- *Lifecycle*: A sequence of Phases that achieve a specific goal. It defines the behavior of a complete process to be enacted in a given project or program.
- *Iteration*: a composite WorkDefinition with a minor milestone.

Artifacts and roles

- *WorkProduct*: This is anything produced, consumed, or modified by a process (document, model, source code, etc.). In the UP this is referred to as an artifact, for example a design model, software development plan, etc.
- *ProcessRole*: A virtual actor, defined by a set of skills, who is responsible for a set of WorkProducts through the execution of a number of Activities

Furthermore, SPEM contains structures (*Package*, *ProcessComponent*, *Discipline*, etc.) that allows to package related process elements into reusable units. This ability facilitates a process engineer to compose a specific development process out of reusable process components.

5.2.2 SPEM v2.0

Currently, OMG is in the process of defining the Software Process Engineering Meta-Model version 2.0 (SPEM2.0) [xvi], which aims at exploiting the UML2.0 features to improve the SPEM v1.1. There are at this moment 2 competing submissions for the SPEM v2.0 standard: one by a consortium of IBM, Adaptive, Fujitsu, ESI and Softeam[xvii], and one by a consortium of Borland, Osellus, MicroSoft and Sun Microsystems [xviii].

Especially, the separation of method contents from their application in the modeling of processes in the submission of the IBM consortium is a promising approach to achieve required reusability and customizability of the framework observing the different real-time embedded system domains of MARTES partners. The idea has been used in [xix] for small projects/teams. A free Java-based reference implementation of SPEM2.0 in EMF2.1 is available from IBM.

SPEM 2.0 separates reusable core method content from its application in processes. Method content provides step-by-step explanations, describing how specific development goals are achieved independently of the placement of these steps within a development lifecycle. Processes take these method elements and relate them into semi-ordered sequences that are customized to specific types of projects [xvii].

In contrast to other method engineering approaches, SPEM 2.0's unique solution allows each process to reference common method guidance from a common method content pool, which then makes up the actual process guidance. Because of these references, changes in the methods will automatically be reflected in all processes using it. However, SPEM 2.0 still allows overwriting certain method related guidance within a process as well as defining individual process-specific relationships for each process element (such as work breakdown and new relations to work products and roles).

Figure 30 shows the difference between method content and process by representing them as two different dimensions. Method content describing how development work, is being

performed, is categorized by disciplines. Each discipline is comprised of tasks (not visible in Figure 30) that provide step-by-step descriptions of how specific development goals are achieved. For a process, tasks have been selected from the method content and placed into workflows ready for instantiation by allocating resources to perform the work and having real work products as the inputs and outputs of the tasks. As a result, the workload graphs shown in Figure 30 can be computed showing work effort for each discipline over time (from left to right).

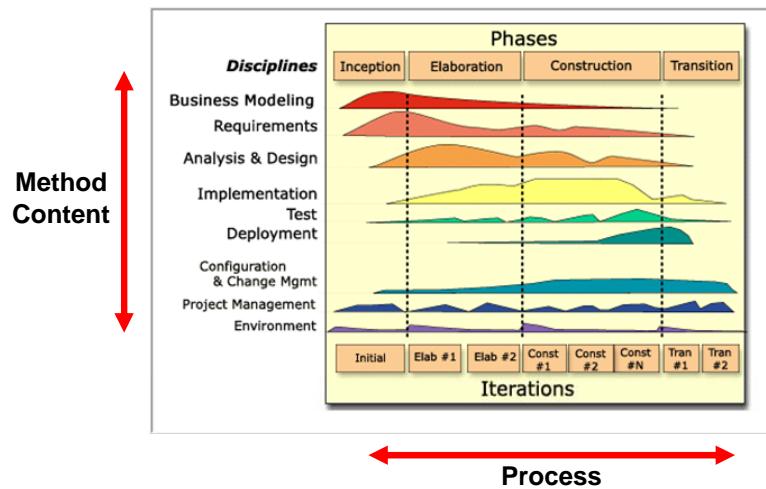


Figure 30. Method Content definition versus the application of Method Content in a Process.

Figure 31 provides an overview of how the key concepts defined in this specification are being positioned to represent method content or process.

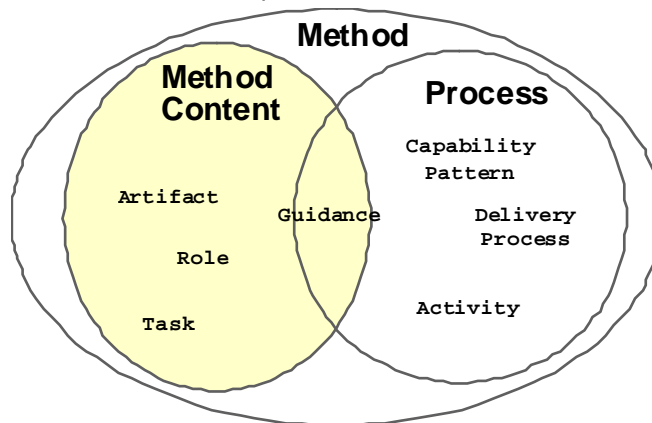


Figure 31. Key terminology defined in this specification mapped to Method Content versus Process.

MARTES	Specification of the Model Driven Engineering Process	Page : 49 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

6 Conclusions

In MDD-oriented development, a considerable part of the total development effort goes into the development of a suitable development environment (MDD Environment or MDDE). Such an environment provides very specific support for the development of one type of applications by offering an integrated set of modeling languages backed with transformations, referred to as infrastructure and a collection of methods that guides the use of the infrastructure.

Building an MDDE requires a considerable amount of effort and requires a large variety of expertise in different domains. Because of this complexity we opted to define a development process that is specifically targeted at developing MDDEs. We choose for a structure that allows the MDDE to be developed somewhat in parallel with the actual application by synchronizing the two processes. In that fashion, application development is not stalled too much by the development of the MDDE. It is obvious that such an approach is only feasible if many other applications of the same type need to be build in the future so that (parts of) the built MDDE can be reused.

Within the MARTES methodology, we propose an MDD Process Framework that contains an initial common MDDE. Three processes are important in this framework:

- The *MDDE process*, which is a generically, UP-based, process. The output of this process is an MDDE that contains an infrastructure part (platforms, languages, transformations, etc.) and a method part (guidelines on how to use the infrastructure).
- The specific *MDDE refinement process* is a more specific version of the MDDE process that guides the partners in creating their own specific instance of the initial, Y-chart, MDDE.
- The original *application process* is the process that guides application development before the introduction of an MDDE, but will need to incorporate the methods offered by the MDDE.

Starting out with a very basic common MDDE and letting each partner specialize into more specific MDDEs seems to be an appropriate first step for MARTES because of the huge variety of partner backgrounds. In future work we will specify the MDDE process in greater detail, including detailed role descriptions, tasks and guidance.

Next, traceability is needed in MDD in order to establish relationships between the different work products in an MDD process. For each project one must choose the right combination of Generic/Full versus Specific Traceability, in the mean time ensuring that both Requirements and Transformation Traceability are covered. We argued for a combination of specific and full/generic traceability. The common part would be specific, addressing only that kind of traceability information that adds additional information to the model that can be reused by the different partners and cannot be derived otherwise. The full/generic part is left to the partners and is not exchanged. The best way to represent the traceability information in the MARTES case is by using the extra-model approach since this can make traceability across different models, currently used by different partners, easier.

MARTES	Specification of the Model Driven Engineering Process	Page : 50 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

A final decision that must be made, which not only applies to traceability, is the technology that will be used to preserve the information. The most convenient way is to select the same technology that is chosen for the other MARTES models.

MARTES	Specification of the Model Driven Engineering Process	Page : 51 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

7 Acronyms

Abbreviation	Definition
CMMI	Capability Maturity Model Integration
MDD	Model-Driven Development
MDDE	Model-Driven Development Environment
MDE	Model-Driven Engineering
OMG	Object Management Group
UP	Unified Process
SPEM	Software Process Engineering Meta-Model

MARTES	Specification of the Model Driven Engineering Process	Page : 52 of 52
		Version: 1.0
		Date: 14/06/2007
		Status: Final
ITEA 04006	Deliverable ID: 1.7	Confid : Public

8 References

-
- [i] Jacobson, I., Booch, G., Rumbaugh, J., The Unified Software Development Process, Addison-Wesley, 1999.
 - [ii] Larman, C., Applying UML and Patterns: An introduction to object-oriented analysis and design and the unified process, 2002.
 - [iii] Ricardo Balduino, Basic Unified Process: A Process for Small and Agile projects, <http://www.eclipse.org/proposals/beacon/>.
 - [iv] Erwin de Kock (ed.), Specification of the Model Transformations (PIM, PSM) in the MARTES methodology, Deliverable report D1.3 of MARTES ITEA 04006 project, Version 1.0, September 2006.
 - [v] Thierry Saunier (ed.), Specification of the Models (PIM, PSM) in the MARTES methodology, Deliverable report D1.2 of MARTES ITEA 04006 project, Version 1.0, June 2006.
 - [vi] Letelier, P., A framework for requirements traceability in uml-based projects, 2002
 - [vii] Pons, C., Kutsche, R.D., Traceability across refinement steps in uml modeling., 2003
 - [viii] Object Management Group: Qvt-merge group submission for mof 2.0 query/view/transformation, 2005, <http://www.omg.org/cgi-bin/apps/doc?ad/2005-03-02>.
 - [ix] Jouault, F., Kurtev, I.: Transforming models with atl, 2005, <http://sosym.dcs.kcl.ac.uk/events/mtip>.
 - [x] Gills, M.: Survey of traceability models in it projects. In: ECMDA-TW Workshop. (2005)
 - [xi] Czarnecki, K., Helsen, S.: Classification of model transformation approaches. In: OOPSLA 2003 WS on Generative Techniques in the context of Model Driven Architecture. (2003)
 - [xii] Jouault, F., Kurtev, I.: Transforming models with atl. In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica (2005)
 - [xiii] Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi, S., Tanca, L.: XML-GL: A graphical language for querying and restructuring XML documents. In: Sistemi Evoluti per Basi di Dati. (1999) 151–165
 - [xiv] Thalheim, B., Thalheim, B.: Entity-Relationship Modeling: Foundations of Database Technology. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2000)
 - [xv] Object Management Group (OMG), Software Process Engineering Metamodel Specification, Version 1.1 (SPEM v1.1), January 2005, <http://www.omg.org/docs/formal/05-01-06.pdf>.
 - [xvi] Object Management Group (OMG), Software Process Engineering Metamodel (SPEM) 2.0 Request For Proposal, <http://www.omg.org/cgi-bin/doc?ad/2004-11-4>.
 - [xvii] Object Management Group (OMG), Software Process Engineering Meta-Model 2.0, ad/2006-06-02, OMG Submission, <http://www.omg.org/cgi-bin/doc?ad/06-06-02>.
 - xviii Object Management Group (OMG), Software Process Engineering Meta-Model 2.0, ad/2006-06-01, OMG Submission, <http://www.omg.org/cgi-bin/doc?ad/06-06-01>.
 - [xix] Ricardo Balduino, Basic Unified Process: A Process for Small and Agile projects, <http://www.eclipse.org/proposals/beacon/>.