EMODE

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

D3.1, Part 1 – Spezifikation der EMODE Entwicklungsumgebung

Telecooperation Report No. 6,
The Technical Reports Series of the Telecooperation Research Division,
TU Darmstadt
ISSN 1864-0516

Written by (in alphabetical order):
Thomas Hamann, Technische Universität Dresden
Gerald Hübsch, Technische Universität Dresden
René Neumerkel, Technische Universität Dresden

DOCUMENT INFORMATION	
TYPE	Deliverable
ID	Deliverable WP3.1 - Spezifikation Entwicklungsumgebung und Ausführungsumgebung
DUE DATE	March 31st 2006
WORK PACKAGE	WP 3 – Technologieentwicklung / Integration
PROJECT	**01ISE02 EMODE** **Enabling Model Transformation-Based Cost Efficient Adaptive Multi-modal User Interfaces**

DOCUMENT STATUS		
ACTION	**BY**	**DATE (dd.mm.yyyy)**
SUBMITTED	Technische Universität Dresden	01.07.2006
WP LEADER	Technische Universität Dresden	21.07.2006
APPROVED	IKV++ Technologies AG	21.07.2006
APPROVED	SAP AG	21.07.2006
APPROVED	Technische Universität Darmstadt	21.07.2006

REVISION HISTORY			
DATE (dd.mm.yyyy)	**VERSION**	**AUTHOR**	**COMMENT**
13.04.2006	0.1	Technische Universität Dresden	Initial draft
09.06.2006	0.2	Technische Universität Dresden	Second draft
30.06.2006	0.9	Technische Universität Dresden	Third draft, Release Candidate 1
21.07.2006	1.0	Technische Universität Dresden	Release of D3.1, Version 1.0

AUTHORS' CONTACT INFORMATION				
NAME	**ORGANISATION**	**EMAIL**	**TEL**	**FAX**
Thomas Hamann	Technische Universität Dresden	thomas.hamann@tu-dresden.de	+49/351/463-3-8380	+49/351/463-3-8251
Gerald Hübsch	Technische Universität Dresden	gerald.huebsch@tu-dresden.de	+49/351/463-3-8380	+49/351/463-3-8251
René Neumerkel	Technische Universität Dresden	neumerkel@rn.inf.tu-dresden.de	+49/351/463-3-8380	+49/351/463-3-8251

Table of Contents

1 INTRODUCTION 4

1.1 OBJECTIVES OF THE DOCUMENT 4
1.2 AUDIENCE 4
1.3 OUTLINE 4
1.4 RELATIONSHIP TO OTHER WORK PACKAGES 4

2 REQUIREMENT ANALYSIS 5

2.1 USE CASES 5
2.1.1 Modeling 5
2.1.2 Model-Transformation 5
2.1.2.1 Specifying transformations definitions 6
2.1.2.2 Preparing a transformation 7
2.1.2.3 Executing transformations 8
2.1.3 Source Code Editing 8
2.1.4 Model/Artifact Management 9
2.2 LIST OF REQUIREMENTS 10
2.2.1 Requirements for the Modeling Tools 10
2.2.2 Requirements for the Transformation Tools 12
2.2.2.1 General Requirements for the Transformation Tools 12
2.2.2.2 Requirements concerning the specification and visualization of transformation definitions/ rules 13
2.2.2.3 Requirements concerning the management of transformation definitions/ rules 16
2.2.2.4 Requirements concerning the execution of transformation definitions 17
2.2.3 Requirements for the Coding Tools 20
2.2.4 Requirements for the Management Tools 21
2.2.5 Overall Requirements for the EMODE tool-chain 23
2.3 LIMITATIONS 23

3 SPECIFICATION OF THE EMODE DESIGN ENVIRONMENT 24

3.1 ARCHITECTURE 24
3.1.1 Overview 24
3.1.2 Modeling Tools 25
3.1.3 Transformation Tools 26
3.1.3.1 Transformation Definition Language 28
3.1.3.2 Transformation Editor 29
3.1.3.3 Transformation Front-end 30
3.1.3.4 Transformation Engine 31
3.1.3.5 Code-Generator 33
3.1.4 Coding Tools 33
3.1.5 Management Tools 33
3.2 PUTTING IT TOGETHER 35
3.2.1 Interaction concepts 35
3.2.2 Using the EMODE tool-chain 36
3.2.3 Development in EMODE 38

4 REFERENCES 39

1 Introduction

Today the development of adaptive, multimodal applications is still a quite complex and expensive task due to the lack of standardized methods and integrated tools. Although there exist well-established methods for the development of user interfaces (e.g. ConcurTaskTrees) and software applications in general (e.g. OOSE), they provide only little to no support for adaptation and multimodality. Therefore, a holistic approach which considers both aspects and also integrates user interface and application development is desired.

The goal of the EMODE project is to provide means for model-driven cost-efficient development of adaptive multimodal applications. The expected solution will consist of a special methodology, a meta-model, as well as a design- and runtime environment for the development and execution of adaptive, multimodal applications. This document deals with the design environment. There exist separate documents for the methodology (D2.1), the EMODE meta-model (D2.2), and the runtime environment (D3.1 part 2).

1.1 Objectives of the Document

The objective of this document is the specification of the design environment for the development of adaptive multimodal applications – also referred to as EMODE tool-chain. In particular, the document specifies the different components of the tool-chain and describes their functionality. Furthermore, the document contains a list of requirements for the EMODE tool-chain on which the specification is based. Last but not least, the document will name the specific components of the tool-chain which actually need to be developed as part of the project.

There will be two more documents (D3.2 and D3.3) concentrating on the model editors and the transformation tools. Therefore, the special requirements/ functionality of these tools will not be fully covered by this document.

1.2 Audience

Partners involved in the development the EMODE tool-chain should refer to this document as a requirements specification. Other audience may find this document useful in order to get an idea of the overall architecture of the EMODE tool-chain.

1.3 Outline

The document is structured as follows. The first part (section 2) is dedicated to the requirements analysis. Here we look at the different use cases in developing adaptive multimodal applications and identify the resulting requirements for the tool-chain. By analyzing the requirements we then come up with a conceptual tool-chain in section 3. We then describe the functionality of the different components of the tool-chain. The last part of the document is about identifying those components of the tool-chain that have to be developed during the project, and those which will be realized by leveraging existing tools.

1.4 Relationship to Other Work Packages

The tool-chain that is specified in this document supports the EMODE methodology described in D2.1. Also, the model editors implement the EMODE meta-model described in D2.2. Using the proposed tool-chain, developers will be able to create adaptive multimodal applications which can then be run on top of the EMODE runtime described in D3.1 part 2.

2 Requirement Analysis

The basis for the specification of the EMODE tool-chain is the identification and analysis of the different requirements. In order to come up with the requirements, we look at the different use cases and characterize the specific tasks and artifacts for model-driven development of adaptive multimodal applications in section 2.1. Based on our findings from the use cases, we come up with a list of requirements in section 2.2

2.1 Use cases

When identifying the use cases for the model-driven development of adaptive multimodal applications we consider the following sources:

- the current state of discussion on the EMODE methodology and meta-models
- the OMG's model driven architecture [1]
- state-of-the-art software development

From the use cases we then derive the requirements for the EMODE tool-chain which are discussed in section 2.2.

2.1.1 Modeling

Since based on the MDA-approach, a great part of the EMODE software development process is dedicated to the creation, manipulation, validation, and visualization of models representing the different views on the software system under development. In EMODE, there will be models for the *goals*, *tasks*, *domain concepts*, *context*, *user interfaces*, *modality*, and *functional core* of adaptive multimodal applications. These models will all be instances of the EMODE meta-model.

When talking about modeling, we also have to consider the following:

- *References between models* – Having models describing different aspects or parts of one software system, we expect that there will be references between different models. For example, elements in the task model might refer to elements in the domain concept model. Also, user interface and functional core will in some way depend on each other.
- *Transformation dependencies* – Apart from referencing each other, models are further connected by transformation traces. For example, a task model could have been generated from a goal model. This leads to the discussion, whether changes in one model should be propagated "up" and "down" the transformation path.
- *Development teams* – It is a common scenario that there are several developers involved in the modeling process, sharing models or even working together on models. In this case we are faced with the problem of synchronizing/ keeping consistent the changes made to the models by different developers.

2.1.2 Model-Transformation

The second focus of the software development in EMODE lies on the transformation of models/ development artifacts into other models/ artifacts. In its model-driven architecture (MDA) the OMG describes the transformation process from a platform-independent model (PIM) to one or more platform-specific models (PSMs) [1]. However, the idea of model-driven software development is not limited to a one-step transformation from one PIM into one or more PSMs [3, S.22]. In fact, model-transformations can be applied at basically any point during a software development process. In EMODE we expect to have transformations at various points, for example when transforming a goal model into an initial task model. The concrete transformations to be used in EMODE are still to be identified in D2.3.

When talking about model-transformation, we find that there exist different types:

- model-to-model transformations
- model-to-code transformations (a.k.a. code generation)
- code-to-model transformations (a.k.a. re-engineering)

In EMODE, we want to concentrate on the first variant, model-to-model transformation. In its most basic form, model-to-model transformation is about transforming one model into another model which is not necessarily based on the same meta-model. More complex scenarios can include merging or splitting of models. Again, the models involved in the transformation do not have to be instances of the same meta-model.

Before we discuss the particular use cases in model-transformation, we need to introduce some definitions:

Model-Transformation – refers to the actual process of generating a set of target models from a set of source models

Transformation definition – formally specifies how to generate a set of target models from a set of source models; contains at least one transformation rule

Transformation rule – formally specifies how one or more instances of a meta-model element are to be transformed into one or more instances of another meta-model element

So far we have talked about model-transformation in general. In the remainder of this section we take a closer look at the different steps in model-transformation:

1. the specification of transformation definitions (section 2.1.2.1)
2. the preparation of a transformation (section 2.1.2.2)
3. the execution of a transformation (section 2.1.2.3)

2.1.2.1 Specifying transformations definitions

The first step in model-transformation is to write a transformation definition. In particular, this means writing appropriate transformation rules that make up such a definition. Transformation rules can generally be defined in two ways:

- *declarative* – transformation rules specify what should be transformed into what without giving details about how this is achieved
- *imperative/programmatic* – transformation rules specify algorithms for transforming model elements into other model elements

Additionally, transformation rules can be defined at different meta-levels according to MOF's four-layer architecture:

- at *M2-level*, also known as *model-type-mapping* [1] – transformation rules are specified upon meta-model elements
- at *M1-level*, also known as *model-instance-mapping* [1] – transformation rules are specified upon model elements

Some clarification:

Transformation definitions/ rules are specified between meta-model elements (M2-level). Besides that, the developer may put so called *marks* on concrete model elements, thereby specifying

- which transformation rule to apply to a concrete model element or
- what target model element to generate from a concrete source model element

When talking about specifying model-transformations we consider the following assumptions:

A1: Transformation definitions can involve (on the input as well as on the output side)

- one or more models
- one or more meta-models

A2: Transformation rules can involve (on the input as well as on the output side)

- one or more model elements
- one or more meta-model elements

A3: Transformation rules can include conditions under which they match a specific set of source model elements. These matching conditions can be understood as being queries or filters for the source models, returning a set of model elements.

A4: Transformation definitions can define parameters for which the developer will define values prior to executing a transformation. [3]

A5: It could be possible for a source model element to match more than one transformation rule; it is not clear, yet, what should be done in such a case; especially the following issues will have to be addressed:

- how to define priorities to transformation rules in case their exist several alternative rules for one model element
- how to define the an order for the application of transformation rules in case their exist dependencies between such rules

2.1.2.2 Preparing a transformation

This step involves setting up the transformation environment and includes

- selecting the models to be transformed
- selecting the transformation definition
- specifying the meta-models for the source models (this is sometimes included in the transformation definition)
- setting the parameters for the transformation
- marking the source models for the transformation

The last step may be used to refine the transformation definition on M1-level and helps guide the transformation process in cases such as:

- the developer does not want to transform all source model elements; in this case he would have to specify which elements he wants to include in the transformation
- there exist alternative rules for a certain source model element; in this case the developer would have to specify which of the rules should be applied
- there exist complementary rules for a certain source model element; in this case the developer would have to specify the order in which the rules are applied

So basically, the developer needs to add further information (marks) to the source models that guide the transformation process. Currently we can imagine different types of marking:

- adding meta-information to individual model elements that will serve as additional input for the transformation rules, especially for the matching of source model elements; the available meta-information depends on what has been specified in the transformation definition
- assign specific transformation rules from a transformation definition to individual source model elements; these manual assignments of rules to model elements would then be recognized by the transformation engine; this includes choosing from a number of alternative transformation rules for a single source model element

Considering that different transformation definitions can be applied to one model, we can imagine to assign marks from different transformation definitions to one model.

2.1.2.3 Executing transformations

The third step is about actually executing transformations. There exist different strategies for accomplishing this:

- S1: the transformation engine iterates over the source models and matches each model element with a transformation rule ("tree-walker")
- S2: the transformation engine iterates over the transformation rules and issues queries on the source models according to the matching conditions ("random access")
- S3: of course a combined approach could also be possible

Sometimes, there will be several models generated from one source model, where bridges between the target models have to be generated; for this, we currently see two possible approaches:

- have one transformation definition which specifies how to simultaneously create multiple target models from a single source model (ATL, MTL can do this)
- instead, one could execute multiple model-transformations sequentially on the same model, creating the different target models

Another issue is about choosing the right order for applying transformation rules to models. This is crucial for complementary transformation rules. Thus, there will be some form of scheduling during the execution of transformations.

Last but not least, user interaction could be required during the transformation process to resolve conflicts that cannot be solved automatically. For example, consider the following situations:

- there exist manual changes in target models which will be lost during the transformation process
- there exist several transformation rules that match a source model element and the transformation engine cannot determine automatically which one to apply

2.1.3 Source Code Editing

Even in model-driven software development it may be necessary to manually edit generated code. This is the case when not all of it can be generated automatically. Also, the developer might want to further refine the generated source code. Therefore, it requires special attention for preserving manual changes in the case where code is being re-generated from the models. Last but not least, the generated source code will have to be compiled in order to run on the target platform.

2.1.4 Model/Artifact Management

In common software development processes (not only MDA) there is usually a variety of artifacts created during the development process, such as models, requirements, source code, etc. Thus, the management of these artifacts is an important task and it involves:

- persistence / storing of artifacts
- versioning
- multi-user access / shared access / sharing of artifacts
- authentication and authorization
- merging / conflict management
- import and export / model exchange

2.2 List of Requirements

From the use cases we derive the following preliminary tool categories:

- modeling tools
- transformation tools
- management tools
- coding tools

The requirements listed here will therefore be assigned to these tool categories.

2.2.1 Requirements for the Modeling Tools

Requirements for the Model Editors

Requirement-ID: **M1**	*Short Name:* **Model Editing**	*Scientific Priority:* *Project Priority:*
Long Name: Support for the creation and manipulation of models		
Description: The Modeling Tools must provide means for the creation and manipulation of models based on the EMODE meta-model.		
Implies: Meta-model support (**M2**)		

Requirement-ID: **M2**	*Short Name:* **Meta-model Support**	*Scientific Priority:* *Project Priority:*
Long Name: Support for the EMODE meta-model		
Description: The Modeling Tools must support the different packages of the EMODE meta-model as defined in WP2. This means, that there could be specialized tools for different meta-model packages – say model types – if necessary. For example, we could have different tools for modeling context, abstract user interfaces, functional core, etc.		

Requirement-ID: **M3**	*Short Name:* **Model Visualization**	*Scientific Priority:* *Project Priority:*
Long Name: Means for visualizing models		
Description: The Modeling Tools must support the notation for the meta-model elements as defined in WP2. This could include graphical as well as textual notation.		

Requirement-ID: **M4**	*Short Name:* **Model Syntax Checking**	*Scientific Priority:* *Project Priority:*
Long Name: Means for validating models		
Description: The Modeling Tools must assure conformance of the models to the EMODE meta-model at any time during development.		

Requirement-ID: **M5**	*Short Name:* **Model Import/Export**	*Scientific Priority:*
		Project Priority:
Long Name: Support for the import and export of models		
Description: The Modeling Tools must provide means for the import and export of models from and to standard formats such as XMI. This is required in order to allow developers to exchange information with other developers not having access to the corporate model management infrastructure.		

Requirement-ID: **M6**	*Short Name:* **Model References**	*Scientific Priority:*
		Project Priority:
Long Name: Support for references between models		
Description: From the current state of discussion on the EMODE meta-model, we expect to have several models of a software system under development, each describing different aspects or parts of that system. Furthermore, we expect that there will be references between the elements of these models. For example, elements in the task model might refer to elements in the domain concept model. As we expect to have separate modeling tools for different parts of the EMODE meta-model – besides providing the developer with means for defining cross-model references – this implies further requirements on the Modeling Tools in order to handle these references.		
Implies: Cross-Model Consistency Checking (**M7**)		

Requirement-ID: **M7**	*Short Name:* **Cross-Model Consistency Checking**	*Scientific Priority:*
		Project Priority:
Long Name: Assuring consistency between models connected by references		
Description: Having references between elements of different models, it must be assured somehow that those references are not broken by editing the individual models. For example, when the developer deletes a model element from model A which is being referenced by an element in model B, then either the reference will have to be updated/deleted or the developer should not be allowed to delete the element from model A in the first place. In order to achieve this, we need some form of change impact analysis.		
Derived from: Model References (**M6**), Transformation Traces (**TE6**)		
Implies: Change Impact Analysis (**M8**)		

Requirement-ID: **M8**	*Short Name:* **Change Impact Analysis**	*Scientific Priority:*
		Project Priority:
Long Name:		
Description: Because of dependencies between models – coming from cross-model references as well as transformation traces – we require so called change impact analysis. This means, that the		

Modeling Tools should be able to compute the consequences of an action the developer is about to perform (like renaming or deleting a model element).
Derived from: Cross-Model Consistency Checking (**M7**)

Requirement-ID: **M9**	*Short Name:* **Team Support**	*Scientific Priority:* *Project Priority:*
Long Name: Support for collaboration scenarios		
Description: The Modeling Tools should provide means for collaboration between developers. This mainly affects model management which is done by the management tools. However, integration with the management tools is required for the Modeling Tools.		
Implies: Integration with Management Tools (**M10**)		

Requirement-ID: **M10**	*Short Name:* **Integration with Management Tools**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Modeling Tools should be closely coupled with the Management Tools to allow persistence, versioning etc. for the models developed.		
Derived from: Team Support (**M9**)		

2.2.2 Requirements for the Transformation Tools

2.2.2.1 General Requirements for the Transformation Tools

Requirement-ID: **T1**	*Short Name:* **Support for MOF-based Meta-models**	*Scientific Priority:* *Project Priority:*
Long Name: Support for any MOF-based meta-models		
Description: The Transformation Tools should be able to handle any MOF-based meta-model, not just the EMODE meta-model. There are at least two reasons for this: (1) the EMODE meta-model might be subject to change and we do not want to adjust our transformation tools every time we make changes to the EMODE meta-model and (2) the users of the EMODE methodology may want to integrate their existing methodology → specifying appropriate transformations is one way to realize this		

Requirement-ID: **T2**	*Short Name:* **Support for Model-Type-Mapping**	*Scientific Priority:* *Project Priority:*
Long Name: Support for Model-Type-Mapping (M2-level transformation definition)		

Description: The Transformation Tools must provide means that allow the definition of transformations on the M2-level (level of meta-model elements). Furthermore, the Transformation Tools must be able to execute transformations defined at the M2-level.

Requirement-ID: **T3**	*Short Name:* **Support for Model-Instance-Mapping**	*Scientific Priority:* *Project Priority:*
Long Name: Support for Model-Instance-Mapping (M1-level transformation definition)		
Description: The Transformation Tools must provide means that allow the definition of transformations on the M1-level (level of model elements). Furthermore, the Transformation Tools must be able to execute transformations defined at the M1-level.		

Requirement-ID: **T4**	*Short Name:* **Integration with other Tools**	*Scientific Priority:* *Project Priority:*
Long Name: Integration of the transformation tools with other development tools		
Description: The Transformation Tools must be connected to the Modeling Tools on one hand and to the Management Tools on the other hand.		

Besides these general requirements, there exist special requirements concerning

- the specification and visualization of transformation definitions/ rules
- the management of transformation definitions/ rules
- the execution of transformations definitions/ rules, including
  - preparation of transformations and models
  - controlling and monitoring of transformations
  - visualization of transformation results
  - conflict management

Note: This section focuses on model-to-model transformation. There are special requirements for model-to-code transformation which are not addressed here. This, however, is not a problem since we do not plan to develop tools for model-to-code transformation, but instead use existing tools.

2.2.2.2 Requirements concerning the specification and visualization of transformation definitions/ rules

Requirements for the Transformation Editor

Requirement-ID: **TS1**	*Short Name:* **Writing Transformation Definitions**	*Scientific Priority:* *Project Priority:*
Long Name: Support for the creation and manipulation of transformation definitions		
Description: The Transformation Tools must provide means for writing transformation definitions based on a transformation meta-model.		

Requirement-ID: **TS2**	*Short Name:* **Syntax Checking and Highlighting**	*Scientific Priority:*
		Project Priority:
Long Name: Support for syntax checking and highlighting when writing transformation definitions/ rules		
Description: The Transformation Tools should provide automatic syntax checking and highlighting to (1) assure the correctness of the transformation definition with respect to the transformation language and (2) to ease the development of transformation definitions.		

Requirement-ID: **TS3**	*Short Name:* **Meta-model Access**	*Scientific Priority:*
		Project Priority:
Long Name: Access to the meta-model(s) when writing transformation definitions/ rules		
Description: Having access to the meta-model(s) when writing transformation definitions/ rules enables the developer to choose from the meta-model definition the elements and properties he wants to use in a concrete model-transformation. Besides that, it reduces – if not eliminates – the cases where the developer accidentally specifies elements and/or properties which do not exist in the corresponding meta-model.		

Requirement-ID: **TS4**	*Short Name:* **Validation**	*Scientific Priority:*
		Project Priority:
Long Name: Support for the validation of transformation definitions/ rules		
Description: The Transformation Tools must provide means for the validation of transformation definitions/ rules. Example: Not only should the transformation definitions conform to the syntax of the transformation language (as assured by the syntax checking), it must also be assured that the meta-model elements are used correctly with respect to the syntax of the meta-model(s).		

Requirements for the Transformation Definition Language

Requirement-ID: **TS5**	*Short Name:* **Multiple Source Models**	*Scientific Priority:*
		Project Priority:
Long Name: Support for multiple source models in transformation definitions		
Description: It must be possible to write transformation definitions which have more than one source model. Each source model could thereby be an instance of a different meta-model.		
Related to: Model Merging (**TS12**)		

Requirement-ID: **TS6**	*Short Name:* **Multiple Target Models**	*Scientific Priority:*
		Project Priority:
Long Name: Support for multiple target models in transformation definitions		
Description: It must be possible to write transformation definitions which have more than one target model.		

Again, each target model could thereby be an instance of a different meta-model.
Related to: Model Splitting (**TS13**)

Requirement-ID: **TS7**	*Short Name:* **Pattern Matching**	*Scientific Priority:* *Project Priority:*
Long Name: Support for pattern matching in transformation rules		
Description: It should be possible to define transformation rules that match a given pattern/set of source model elements rather than just one source model element *at a time*. Each element of a pattern can be an instance of a different meta-model element.		

Requirement-ID: **TS8**	*Short Name:* **Pattern Generation**	*Scientific Priority:* *Project Priority:*
Long Name: Support for pattern generation in transformation rules		
Description: It should be possible to define transformation rules that generate a given pattern/set of target model elements rather than just one target model element *at a time*. Again, each element of that pattern can be an instance of a different meta-model element.		

Requirement-ID: **TS9**	*Short Name:* **Parameters**	*Scientific Priority:* *Project Priority:*
Long Name: Support for parameters in transformation definitions		
Description: Parameters allow the developer to tune the transformation definition to his needs. On the other hand, it allows others to write more generic/reusable transformations.		

Requirement-ID: **TS10**	*Short Name:* **Filter Expressions/ Queries**	*Scientific Priority:* *Project Priority:*
Long Name: Support for filter expressions/ queries in transformation definitions		
Description: Filter expressions and/or queries are required in order to • specify which source model elements a transformation rule should match • retrieve information from source and target models to be used by transformation rules		

Requirement-ID: **TS11**	*Short Name:* **Reuse of Transformation Rules**	*Scientific Priority:* *Project Priority:*
Long Name: Support for the import of existing transformation rules		
Description: There must be a way for reusing existing transformation rules when writing transformation definitions.		

Requirement-ID:	*Short Name:*	*Scientific Priority:*

TS12	**Model Merging**	*Project Priority:*
Long Name:		
Description: It must be possible to specify transformations which generate a combined target model from a set of source models.		

Requirement-ID: **TS13**	*Short Name:* **Model Splitting**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: It must be possible to specify transformations which generate a set of target models from a source model. This includes the generation of necessary bridges between the generated target models.		

Requirement-ID: **TS14**	*Short Name:* **Model-Type-Mapping**	*Scientific Priority:* *Project Priority:*
Long Name: Support for Model-Type-Mapping		
Description: It must be possible to define transformation rules on the M2-level (at the level of meta-model elements)		

Requirement-ID: **TS15**	*Short Name:* **Model-Instance-Mapping**	*Scientific Priority:* *Project Priority:*
Long Name: Support for Model-Instance-Mapping		
Description: It must be possible to define transformation rules on the M1-level (level of model elements)		

Requirement-ID: **TS16**	*Short Name:* **Declarative Language**	*Scientific Priority:* *Project Priority:*
Long Name: Use of a declarative transformation language		
Description: The Transformation Tools should provide a declarative transformation definition language for writing transformation definitions.		

2.2.2.3 Requirements concerning the management of transformation definitions/ rules

Requirements for the Transformation Repository

This is covered by the requirements for the Management Tools.

Requirement-ID: **TM1**	*Short Name:* **Persistence of Transformation Definitions**	*Scientific Priority:* *Project Priority:*

Long Name:
Description: The Transformation Tools must provide means for keeping transformation definitions persistent.

Requirement-ID: **TM2**	*Short Name:* **Access to Transformation Definitions**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: To make transformation rules available for developers and transformation tools they should be stored in a central repository		

2.2.2.4 Requirements concerning the execution of transformation definitions

Requirements for the Transformation Engine and the Transformation Front-end

Requirement-ID: **TE1**	*Short Name:* **Setting up Transformations**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Transformation Tools must support the developer in setting up a transformation. This includes: • selecting the desired transformation definition • selecting the models for the transformation • preparing the models for the transformation / marking the models • setting the required parameters for the transformation		
Implies: Marking of Source Models (TE2)		

Requirement-ID: **TE2**	*Short Name:* **Marking of Source Models**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Transformation Tools should give the developer the opportunity to add meta-information to the source models in order to guide the transformation process. This activity is referred to as *marking the models* or *model-instance-mapping* and includes: • selecting the model elements to be transformed • adding special meta-information that is being used as input by some transformation rule or affects the matching of source model elements with transformation rules (e.g. special stereotype settings that are evaluated in the process of transformation) • assigning specific transformation rules to source model elements in order to o choose from alternative rules matching the element o specifying a rule in the case the element matches no rule		
Related to:		

Model-Instance-Mapping (TS15)

Requirement-ID: **TE3**	*Short Name:* **Executing Transformations**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Transformation Tools must provide means for executing transformation rules specified in transformation definitions. In general, there are two ways to achieve this: (1) transformation rules could be interpreted by a special processing unit or (2) the transformation definitions could be compiled and run as stand-alone programs. Either way, the part actually doing the transformation must be able to: • access the source and target models • match source model elements and transformation rules • apply filters to the source and target models • issue queries against the source and target models • iterate over the elements in the source and target elements • create or modify elements in the target models • create or modify transformation traces • access the meta-models (e.g. in order to identify generalization relations between meta-model elements and to ensure conformity of the generated target models) • create and maintain a list of the transformation results • report any errors that occurred • resolve conflicts that may arise during the transformation • report the status/ progress of the current transformation		
Implies: Access to Models (**TE4**), Access to Meta-models (**TE5**), Transformation Traces (**TE6**), Resolving Conflicts (**TE7**), Reporting (**TE8**)		

Requirement-ID: **TE4**	*Short Name:* **Access to Models**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Transformation Tools responsible for realizing the transformation must have access to the source models as well as the target models. This includes the ability to create target models in the first place.		

Requirement-ID: **TE5**	*Short Name:* **Access to Meta-models**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Transformation Tools responsible for realizing the transformation must have access to the meta-models of the models involved in the transformation process in order to • assure the correctness of the transformation rules • assure the conformity of the (target) models with the meta-models • set correct default values for the model element properties which are not explicitly		

specified in the transformation rule • to reflect on the dependencies between different meta-model elements

Requirement-ID: **TE6**	*Short Name:* **Transformation Traces**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Transformation Tools must keep a record of every transformation, holding information on what has been transformed into what using what rule. This is required in order to accomplish cross-model-consistency checks (**M7**)		

Requirement-ID: **TE7**	*Short Name:* **Resolving Conflicts**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Transformation Tools must provide means for resolving conflicts that may occur during a transformation, for example • if multiple transformation rules match a source model element and the system cannot choose the correct rule automatically • if there exist manual changes in the target models which would be lost during the process of re-generating target models In order to be able to resolve conflicts, the transformation tool in charge of the transformation should therefore o report that there is a conflict and o ask for specific user input to solve the conflict if possible		
Implies: User Interaction (**TE9**)		

Requirement-ID: **TE8**	*Short Name:* **Reporting Transformation Results**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: Once a transformation has finished, the Transformation Tools should present to the developer the results of that transformation, listing: • what has been transformed into what using which rule (→ transformation traces) • element that could not be transformed • conflicts that occurred during the process of transformation and whether or not they could be resolved • errors in the transformation definition that have been detected during the process of the transformation • any other errors that occurred during the process of transformation		

Requirement-ID: **TE9**	*Short Name:* **User Interaction**	*Scientific Priority:* *Project Priority:*
Long Name:		

Description: The Transformation Tools should provide means for requesting user input during the process of transformation. This is needed, for example, to resolve conflicts occurring during a transformation.
Derived from: Resolving Conflicts (**TE7**)

Requirement-ID: **TE10**	*Short Name:* **Source Model/ Target Model Relations**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Transformation Tools should not only allow the generation/transformation of target models from source models but also they should allow for testing whether a set of source and target models comply with a given transformation definition.		

2.2.3 Requirements for the Coding Tools

Requirements for the IDE

Requirement-ID: **C1**	*Short Name:* **State-of-the-Art Coding Tools**	*Scientific Priority:* *Project Priority:*
Long Name: Provide the functionality of state-of-the-art IDE's		
Description: In EMODE we also require the functionality of source code editors, compilers, debuggers, and other SOTA development tools.		

Requirement-ID: **C2**	*Short Name:* **Team Support**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Coding Tools should provide means for collaboration between developers. This is already accomplished by the Management Tools. Therefore, integration of the Coding Tools with the Management Tools is required.		
Implies: Integration with Management Tools (**C3**)		

Requirement-ID: **C3**	*Short Name:* **Integration with Management Tools**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description: The Coding Tools must be coupled with the Management Tools.		
Derived from:		

Team Support (**C2**)

2.2.4 Requirements for the Management Tools

Requirements for the Repositories for models, source code, transformation definitions and traces

Requirement-ID: **MM1**	*Short Name:* **Persistency**	*Scientific Priority:* *Project Priority:*
Long Name: Persistency of models, source code, transformation definitions, and traces		
Description: The Management Tools must provide means for storing the different artifact types in EMODE, which include models, source code, transformation definitions, and transformation traces.		

Requirement-ID: **MM2**	*Short Name:* **Versioning**	*Scientific Priority:* *Project Priority:*
Long Name: Versioning of models, source code, transformation definitions, and traces		
Description: The Management Tools must provide means for keeping and organizing different versions of the EMODE artifacts, including models, source code, transformation definitions, and traces.		

Requirement-ID: **MM3**	*Short Name:* **Syntax Checking**	*Scientific Priority:* *Project Priority:*
Long Name: Validation of models, source code, transformation definitions, and traces		
Description: The Management Tools must assure that the stored artifacts conform to their corresponding meta-models.		

Requirement-ID: **MM4**	*Short Name:* **Collaboration**	*Scientific Priority:* *Project Priority:*
Long Name: Support for collaboration scenarios		
Description: Developers working together on a project should be able to share the various artifacts (models, source code, transformation definitions, transformation traces). The Management Tools must provide means for shared access to the artifacts. Furthermore, there should be mechanisms for import and export of artifacts using standard formats.		
Implies: Shared Access (**MM5**)		

Requirement-ID: **MM5**	*Short Name:* **Shared Access**	*Scientific Priority:* *Project Priority:*
Long Name: Shared access to models, source code, transformation definitions, and traces		
Description:		

Developers must have shared access to the models, source code, transformation definitions, and traces. At this point, we must decide whether we want to support exclusive or concurrent access. The former requires some synchronization mechanism (e.g. a checkout-mechanism) while the latter requires the ability to merge changes. In both cases, access to the artifacts must be authorized first.
Derived from: Collaboration (**MM4**)
Implies: Synchronization (**MM6**) **or** Merging (**MM7**), Authorization (**MM8**)

Requirement-ID: **MM6**	*Short Name:* **Synchronization**	*Scientific Priority:* *Project Priority:*
Long Name: Synchronize access to models, source code, transformation definitions, and traces		
Description: The Management Tools must prohibit concurrent changes to models, source code, transformation definitions, and traces.		
Derived from: Shared Access (**MM5**)		
Conflicts: Merging (**MM7**)		

Requirement-ID: **MM7**	*Short Name:* **Merging**	*Scientific Priority:* *Project Priority:*
Long Name: Merging of concurrent changes to models, source code, transformation definitions, and traces		
Description: The Management Tools must provide the ability to merge concurrent changes from developers to models, source code, transformation definitions, and traces.		
Derived from: Shared Access (**MM5**)		
Conflicts: Synchronization (**MM6**)		

Requirement-ID: **MM8**	*Short Name:* **Authorization**	*Scientific Priority:* *Project Priority:*
Long Name: Access to models, source code, transformation definitions, and traces must be authorized		
Description: The Management Tools must provide means for authorizing developers before they are allowed to access models, source code, transformation definitions, and traces.		
Derived from: Shared Access (**MM5**)		

Requirement-ID: **MM9**	*Short Name:* **Integration with other tools**	*Scientific Priority:* *Project Priority:*
Long Name:		
Description:		

The management tools must integrate with the rest of the development tools.

2.2.5 Overall Requirements for the EMODE tool-chain

Requirement-ID:	*Short Name:* **Integration with existing SDP**	*Scientific Priority:* / *Project Priority:*
Long Name: Integration with existing software development processes		
Description: The EMODE development tools should provide means that allow the integration of existing software development processes into the EMODE methodology. This could be achieved, for example, by providing means for importing existing artifacts from other SDPs and also vice versa.		

2.3 Limitations

L1 The EMODE tool-chain does not support code-to-model-transformation.

3 Specification of the EMODE Design Environment

In this section we discuss the architecture for the EMODE design environment which we derived from the above requirements. In section 3.2 we discuss the interaction between the different components of the architecture and describe the intended usage of the tools. Also, we identify the architecture components to be developed in EMODE.

3.1 Architecture

In this section we introduce the architecture for the EMODE design environment. We start with an overview of the whole architecture and the described the different components in detail.

3.1.1 Overview

From the use cases described in section 2.1 and the requirements listed in section 2.2, it becomes obvious that we need tool support for the following design tasks:

- creating and manipulating models
- writing and executing transformation definitions
- generating and manipulating source code
- managing models, source code, transformation definitions, and transformation traces

Figure 1 shows the proposed design environment for EMODE. It consists of several components which can be divided into the following categories:

- modeling tools
- transformation tools
- coding tools
- management tools

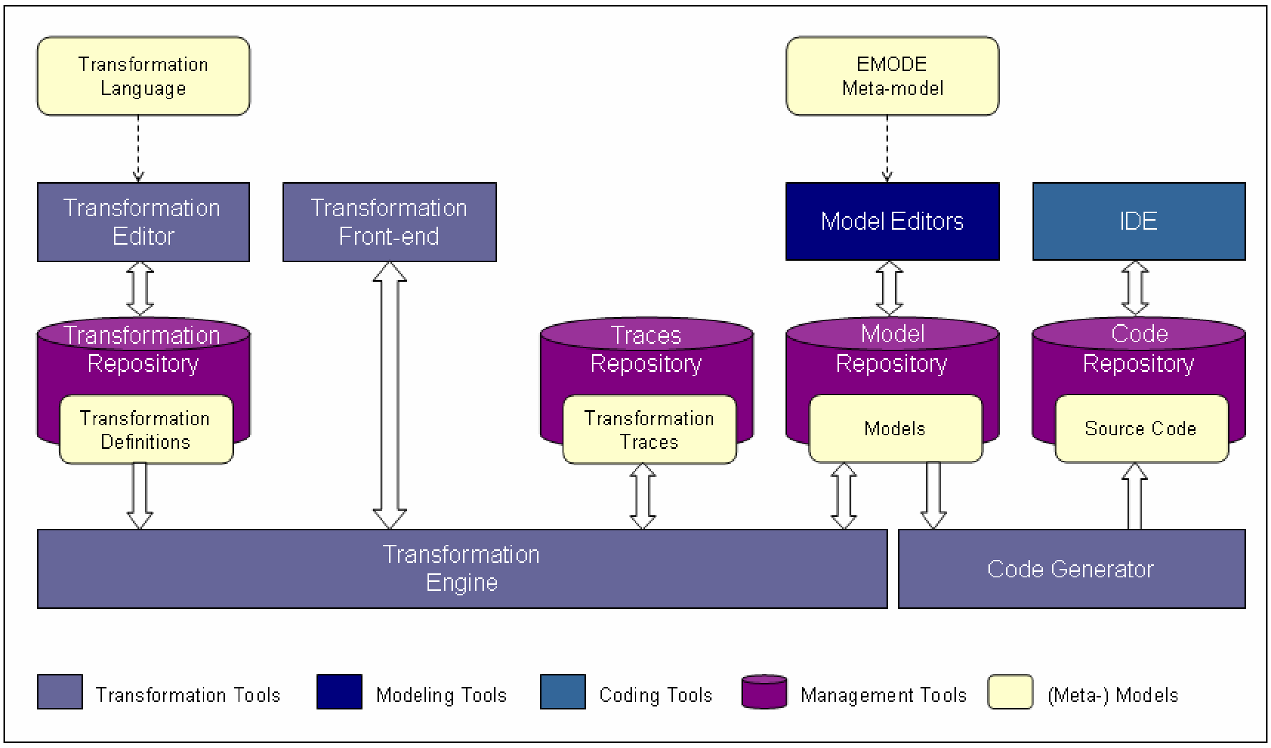

Figure 1 Design Environment for EMODE

In the remainder of this section, we explain the different components of the EMODE tool-chain in more detail.

3.1.2 Modeling Tools

This category consists of the *Model Editors* for the different model types in EMODE as defined in D2.2. These model types include goal models, task models, domain concept models, as well as models describing modalities, abstract user interfaces (AUI), and functional core adapters. For an explanation of the different model types please refer to D2.2. The allocation of model editors to model types, as well as the editors themselves, will be discussed in D3.2. According to the current state of discussion, we will have at least the following editors:

- an editor for goal models, task models, functional core adapters
- an editor for domain concept models
- an editor for modality models and abstract user interface models

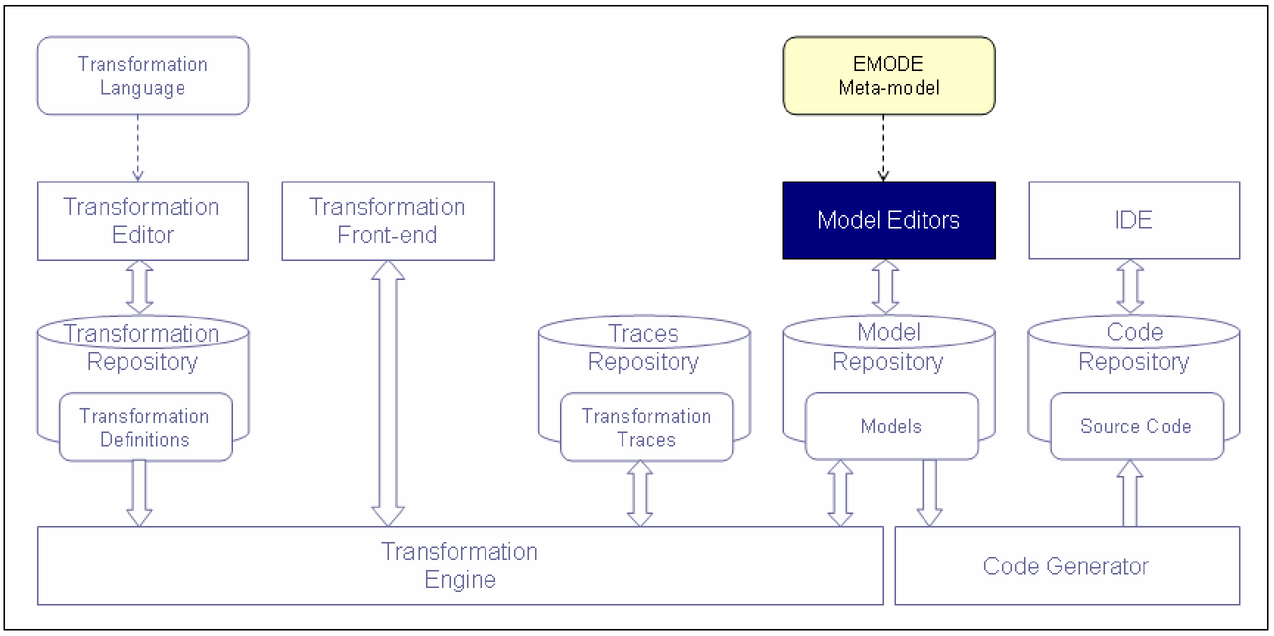

Figure 2 Modeling Tools

We suggest having specialized model editors because we think that the different model types will require different visualization and usability concepts. For example, abstract user interface design will require more sophisticated tool support as, for example, goal modeling. However, the model editors will also provide some common functionality which is described in the following. Details on the different model editors will be provided in D3.2.

Function-ID: **FM1**	*Name:* **Support for the EMODE meta-model**	*Requirement-ID:* **M2**
Description: The model editor will provide mechanisms which allow modeling of the concepts defined in the EMODE meta-model.		

Function-ID: **FM2**	*Name:* **Support for graphical model editing**	*Requirement-ID:* **M1, M3**
Description: Since the EMODE meta-model will provide a graphical notation for its concepts, the model editors will include the necessary functionality for the manipulation of models using diagrams		

consisting of D-shapes.

Function-ID: **FM3**	*Name:* **Support for model validation**	*Requirement-ID:* **M4**
Description: The model editors will provide means which make sure that all the produced models conform to the EMODE meta-model. For this, there exist at least two general approaches: 1. the model editors could dynamically adjust the set of possible user actions in order to allow only those actions which do not result in a violation of the meta-model rules 2. the model editors could – at certain points in time – compare the current model against the meta-model, notifying the user of any conflicts		

Function-ID: **FM4**	*Name:* **Support for model import/export**	*Requirement-ID:* **M5**
Description: The model editors will allow the import of models from the following formats: • XMI, … The model editors will allow the export of models into the following formats: • XMI, …		

Function-ID: **FM5**	*Name:* **Integration with Management Tools**	*Requirement-ID:* **M9**, **M10**
Description: The model editors will work directly on the repositories, instead of files.		

Function-ID: **FM6**	*Name:* **Cross-Model-Consistency-Checking**	*Requirement-ID:* **M6**, **M7**
Description: The model editors will check consistency between models that have references between their elements.		

Function-ID: **FM7**	*Name:* **Change-Impacts-Analysis**	*Requirement-ID:* **M8**
Description: The model editors will provide mechanisms which allow the calculation of the expected results of a certain user action prior to the actual execution of that action. These results will be shown to the user when he is about to perform a certain action, e.g. deleting a model element. This is related to the cross-model-consistency-checking described above.		

3.1.3 Transformation Tools

The tools in this category provide an infrastructure for the specification and execution of model-transformations.

There exist different approaches to model-to-model-transformation. Czarnecki and Helsen [1], for example, distinguish between:

- direct manipulation approaches (imperative, algorithms)

- relational approaches (declarative, mapping rules)
- graph-transformation-based approaches (mostly declarative, graph patterns)
- structure-driven approaches
- hybrid approaches (imperative and declarative)

A different classification is given by Jézéquel in [4]. He categorizes current model transformation techniques as follows:

- general purpose programming languages
- generic transformation tools (e.g. XSLT [6], graph transformations)
- scripting languages included in CASE tools
- dedicated model transformation tools/ languages (e.g. QVT Relations[7])
- model-driven approach on transformations (e.g. Kermeta [8])

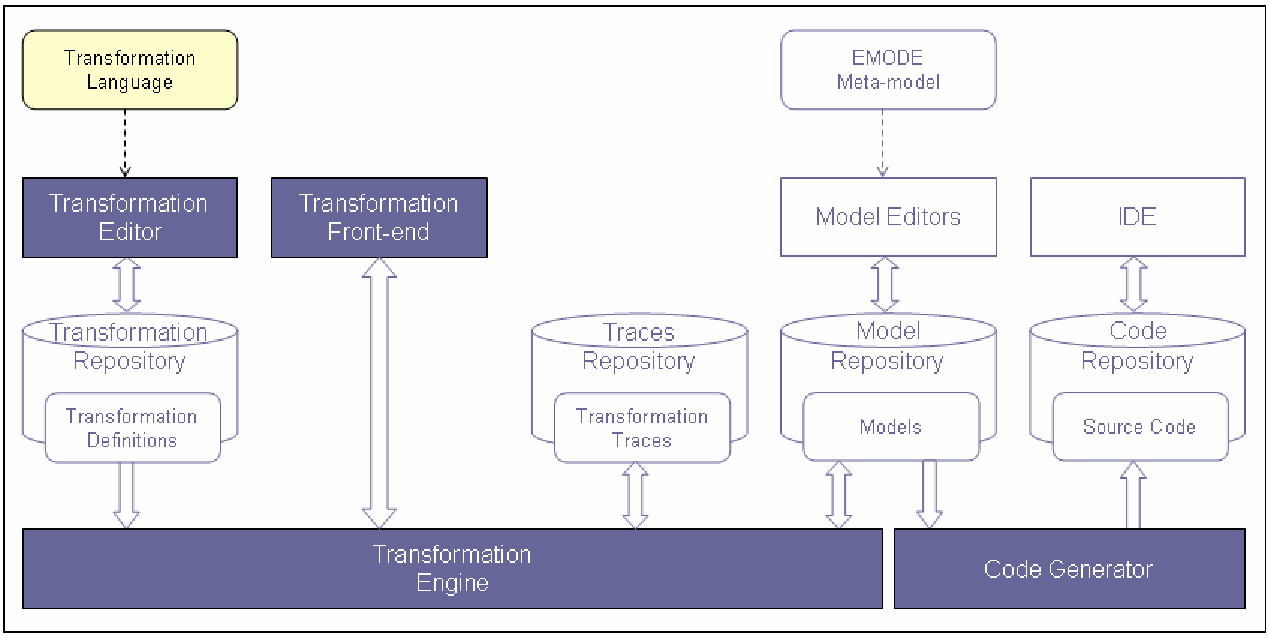

Figure 3 Transformation Tools

The transformation tools in the EMODE design environment will follow an approach similar to the QVT Relations as defined in the OMG's QVT specification [7]. Following this approach, a transformation definition will consist of a number of transformation rules which are interpreted and executed by a transformation engine.

The resulting transformation tools are shown in Figure 3. They include the

- *Transformation Definition Language* which defines a grammar for writing transformation definitions
- *Transformation Editor* which supports the developer in writing transformation definitions using the transformation definition language
- *Transformation Engine* which executes the transformation definitions written in the transformation definition language
- *Transformation Front-end* which is used for preparing and controlling the execution of transformation definitions
- *Code Generator* which allows transformation of models into code; this is different to the approach used for model-to-model transformation above

In the following, the different transformation tools are explained in detail.

3.1.3.1 Transformation Definition Language

Using this language, developers will be able to define model-transformations on the basis of the EMODE meta-model. More precisely, developers will be able to specify rules that describe mappings between elements of the EMODE meta-model. The transformation definitions can then be used to transform concrete models given that they are instances of the EMODE meta-model. This is also illustrated in Figure 4.

Considering the requirements in section 2.2.2.1 and 2.2.2.2, a transformation definition would typically consist of:

- a set of variables representing the source and target model instances of a transformation
- a set of variables representing the meta-models associated with the transformation
- a set of transformation rules specifying how the source model-elements are to be transformed into target model elements
- a set of parameters which can be used to customize a transformation prior to the execution of such a transformation
- a set of marks which can be assigned to source models prior to the execution of a transformation, allowing model-dependent transformations

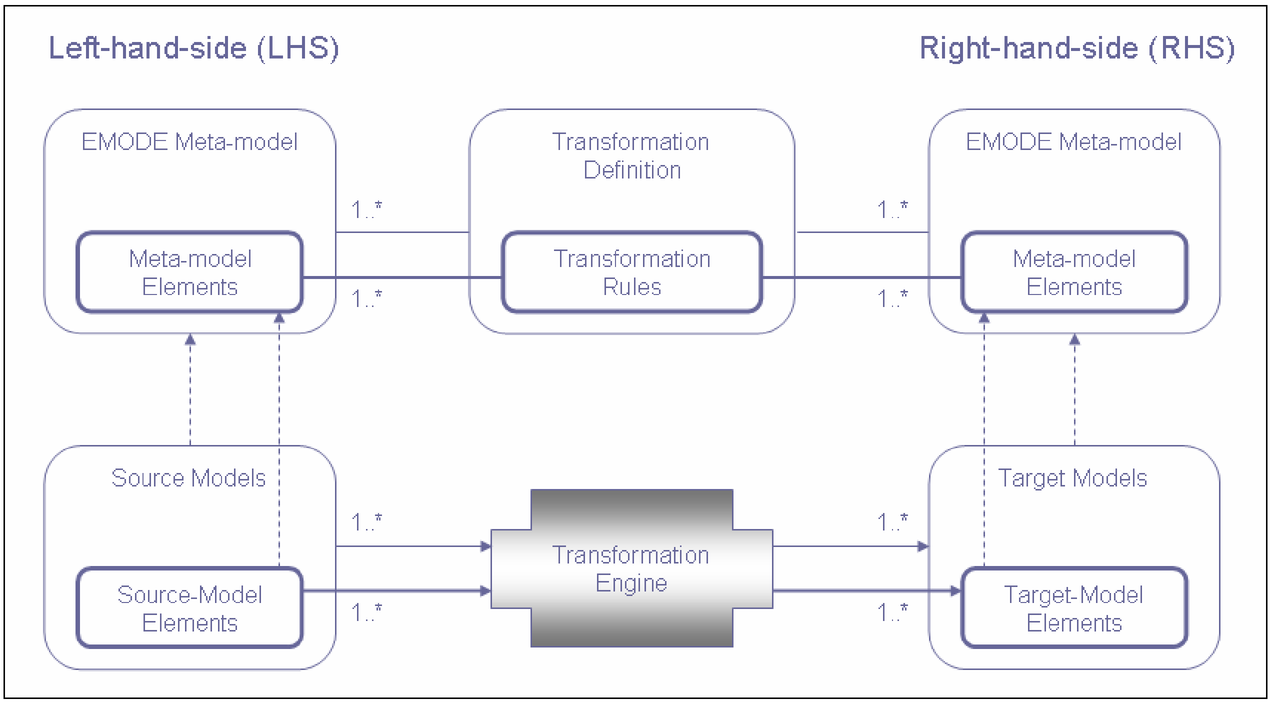

Figure 4 Definition of model-transformations in EMODE

Special attention must be given to the transformation rules, as they define how the source-model elements are to be transformed into target-model elements. Again, based on the requirements in section 2.2.2, we suggest that a transformation rule should consist of:

- on the "left-hand-side"
    - a *matching expression* which selects the set of source model elements to which the rule applies to; this is basically a filter or query expression

  - a *source condition* which further restricts when a certain rule should be executed; these source conditions are evaluated upon
    - transformation parameters
    - information coming from source model or source model elements
    - information coming from target model or target model elements
  - and use filter/ query expressions (e.g. expressed in OCL)
- on the "right-hand-side"
  - one or more *instantiation expressions* which create instances of model elements in the target model; alternatively this could be a *refinement expression*, in the case the target model element already exists
  - both can be prefixed by a *guard condition* which controls when such an expression should be evaluated
  - following an instantiation expression there could be an *initialization block* where the newly created model element instance is initialized; values for the initialization could originate from
    - the source model or source model elements
    - the target model or target model element
    - the transformation parameters
  - following a refinement expression there must be a *refinement block* where the target model element is being refined; as before, the information used to refine the model element can come from
    - the source model or source model elements
    - the target model or target model element
    - the transformation parameters
  - initialization and refinement blocks can make use of *conditional statements* as well as filter/ query expressions

3.1.3.2 Transformation Editor

This is basically an editor for the transformation definition language discussed in section 3.1.3.1. It provides the developer with an interface that supports the authoring of transformation definitions using textual notation. It also provides the following functionality:

Function-ID: **FTS1**	*Name:* **Support for writing transformation definitions**	*Requirement-ID:* **TS1**
Description: This includes basic functionality of a text-based editor.		

Function-ID: **FTS2**	*Name:* **Support for syntax checking**	*Requirement-ID:* **TS2**
Description: The editor will provide syntax highlighting and syntax checking "as you type".		

Function-ID: **FTS3**	*Name:* **Meta-model access**	*Requirement-ID:* **TS3**
Description: The editor will provide a graphical interface to the meta-models. This way, the developer will be able to choose from the available meta-model elements and their properties while writing transformation rules.		

Function-ID: **FTS5**	*Name:* **Validation of transformation rules**	*Requirement-ID:* **TS4**
Description: The editor will provide special mechanisms to validate whether the transformation definitions conform to the meta-model rules. For example, the editor should warn the user when he tries to set a property on a meta-model element which does not exist in the meta-model.		

Function-ID: **FTS6**	*Name:* **Integration with Management Tools**	*Requirement-ID:* **T4**
Description: The editor will store the transformation definitions using in the management tools described in section 3.1.5.		

3.1.3.3 Transformation Front-end

The transformation front-end provides the user interface to the transformation engine. It can be a stand-alone tool or come in form of a plug-in for the model editors. The developer will use the front-end to set up the transformations and models prior to the execution of the transformation itself. This way he will be able to control the transformation process. Finally, the transformation front-end is also used for displaying the results of a transformation. The functionality of the transformation front-end can be derived from the requirements in section 2.2.2.4 as follows.

Function-ID: **FTF1**	*Name:* **Selecting the transformation definition**	*Requirement-ID:* **TE1**
Description: The Transformation Front-end will provide means that allow the developer to select a transformation definition from the list of available definitions.		

Function-ID: **FTF2**	*Name:* **Selecting the source models**	*Requirement-ID:* **TE1**
Description: The Transformation Front-end will provide means that allow the developer to select the source models for the transformation.		

Function-ID: **FTF3**	*Name:* **Setting transformation parameters**	*Requirement-ID:* **TE1**
Description: The Transformation Front-end will provide means that allow the developer to set the concrete values for the specific transformation parameters.		

Function-ID: **FTF4**	*Name:* **Preparing the models (Marking)**	*Requirement-ID:* **TE2**
Description: The Transformation Front-end will provide means that allow the developer to add further meta-information to the models that are used to refine the transformation definition.		

Function-ID: **FTF5**	*Name:* **Reporting transformation results**	*Requirement-ID:* **TE8**

Description: The Transformation Front-end will provide means that allow the developer to view the results of the transformation. In particular, this includes a list of • transformation traces (what has been transformed into what by which rule) • elements that were not transformed • errors that occurred during the process of transformation

3.1.3.4 Transformation Engine

This component is responsible for interpreting and executing the transformation rules written in the transformation definition language.

The transformation engine will require as input:

- the set of source models to be transformed
- the transformation definition consisting of the transformation rules
- the meta-models for the source and target models
- the transformation parameters
- the marks for the source models

The transformation engine will then generate the following output:

- the resulting target models
- trace information about what has been transformed into what using what rule
- a protocol of the transformation (including any errors)

This is summarized in Figure 5. Besides this, the transformation engine will provide the following functionality:

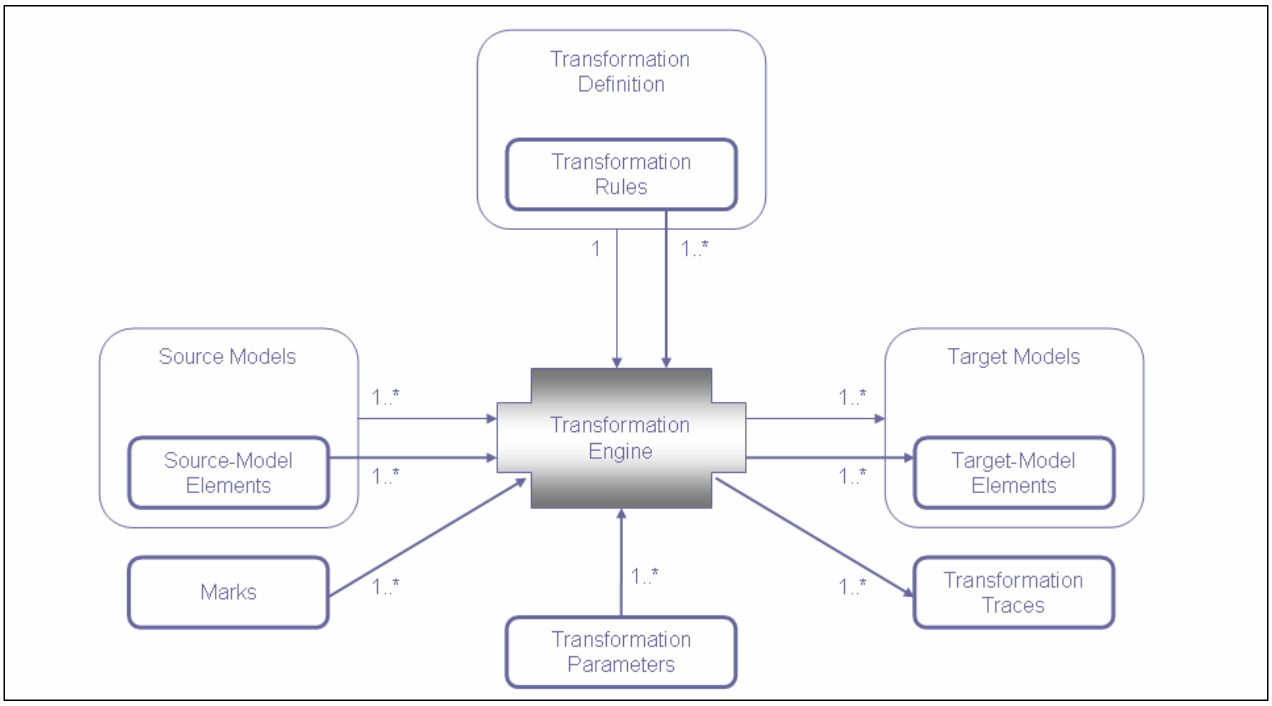

Figure 5 Execution of transformation rules

Function-ID:	*Name:*	*Requirement-ID:*
FTE1	**Operations on source and target models**	**TE3**

Description:
The Transformation Engine will support the following operations on the source and target models:

- matching of model elements and transformation rules
- application of filters to the models, returning a subset of model-elements
- execution of queries against models, returning a subset of model-elements
- iteration over model elements
- creation of new model elements
- modification of existing model elements
    - model elements that have existed prior to the transformation
    - model elements that have been created during the same transformation

Function-ID:	*Name:*	*Requirement-ID:*
FTE2	**Access to repositories**	**TE4**

Description:
Since the source and target models, the transformation definitions and traces are stored in repositories, the Transformation Engine will include the necessary functionality for accessing those repositories.

Function-ID:	*Name:*	*Requirement-ID:*
FTE3	**Scheduling of transformation rules**	

Description:
Sometimes the order, in which the transformation rules are executed, is important. Therefore, the transformation engine will incorporate a strategy for deciding:

- the order in which transformation rules are to be applied
- whether or not multiple transformation rules can be applied to one model element
- how a dependency between two transformation rules can be resolved

Function-ID:	*Name:*	*Requirement-ID:*
FTE4	**Creation of transformation traces**	**TE6**

Description:
As a result of the transformation, the Transformation Engine will also generate transformation traces that hold the information about what concrete source model elements have been transformed into which target model elements, the transformation rule that was used, as well as any parameters that had an impact on the transformation.

Function-ID:	*Name:*	*Requirement-ID:*
FTE5	**Reflection on the meta-models**	**TE5**

Description:
Furthermore, the transformation engine will have access to the meta-models belonging to the source and target models. This way, the transformation engine can evaluate the structure and the dependencies of certain meta-model elements, and therefore:

- decide on the application of certain transformation rules
- assign default values for properties of the model-elements which are not addressed by the transformation rules
- check the resulting models for conformity with the meta-models

Function-ID: **FTE6**	*Name:* **Generation of transformation reports**	*Requirement-ID:* **TE8**
Description: The transformation engine will generate a report on every transformation.		

Function-ID: **FTE7**	*Name:* **Progress feedback**	*Requirement-ID:*
Description: The transformation engine will give immediate feedback on the transformation progress.		

Function-ID: **FTE8**	*Name:* **Source/ Target model relation checking**	*Requirement-ID:* **TE9**
Description: The transformation engine will support a special mode that enables the developer to check whether a set of models comply with a given transformation definition.		

Further details on the transformation engine will be provided in D3.3.

3.1.3.5 Code-Generator

The approaches for model-to-model transformations described above turn out to be impractical for model-to-code transformations. Therefore, there will be a separate component for this in EMODE. It will provide the functionality necessary for transforming models into code. We will not further specify this component here. Instead, we plan to use existing technology and tools for this task in EMODE.

3.1.4 Coding Tools

In addition to model editors and transformation tools we will have tools for editing and compiling the source code that has been generated. These tools will provide the functionality of state-of-the-art coding tools (Req. **C1**). Additionally, they will be connected to a central repository (Req. **C3**) and also provide support for team development scenarios (Req. **C2**).

3.1.5 Management Tools

The functionality for managing the artifacts of the EMODE development process is provided by a set of repositories shown in Figure 6. There will be special repositories for models, source code, transformation definitions, and transformation traces.

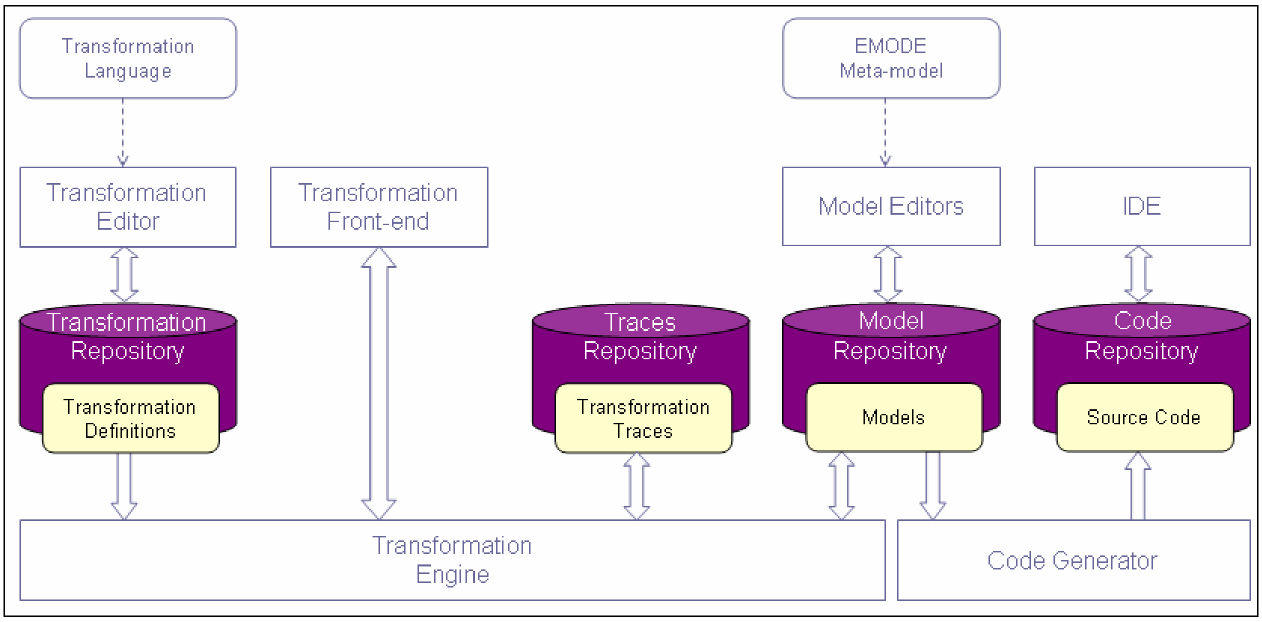

Figure 6 Management Tools

Following is a list of the common functionality for all repositories.

Function-ID: **FMM1**	*Name:* **Persistency**	*Requirement-ID:* **MM1**
Description: The Management Tools will provide means for keeping persistent copies of all artifacts created during the EMODE development process.		

Function-ID: **FMM2**	*Name:* **Versioning**	*Requirement-ID:* **MM2**
Description: The Management Tools will provide means for keeping different versions of persistent copies of artifacts created during the EMODE development process.		

Function-ID: **FMM3**	*Name:* **Syntax checking**	*Requirement-ID:* **MM3**
Description: The Management Tools will automatically check for any artifact stored in the repositories that it conforms to the corresponding meta-model. In this case, the EMODE meta-model.		

Function-ID: **FMM4**	*Name:* **Shared access**	*Requirement-ID:* **MM5**
Description: The Management Tools will provide means which coordinate multi-user access to the artifacts stored in the repositories. They will realize one of the following strategies: • synchronized access (checkout and locking mechanisms) • concurrent access (merge algorithms)		

Function-ID: **FMM5**	*Name:* **Authorization**	*Requirement-ID:* **MM7**

Description: The Management Tools will provide means which allow for authorization of users prior to accessing the repositories.

Function-ID:	*Name:*	*Requirement-ID:*
FMM6	**Integration with other tools**	**MM8**
Description: The Management Tools will provide an API that allows the integration with the model editors, transformation tools and coding tools.		

Function-ID:	*Name:*	*Requirement-ID:*
FMM7	**Support for MOF-based meta-models**	
Description: The Management Tools will provide the above functionality to any artifact that is an instance of a meta-model which in turn is an instance of the MOF meta-model.		

Function-ID:	*Name:*	*Requirement-ID:*
FMM8	**Administration tools**	
Description: The Management Tools will also include tools that allow administration of the repositories.		

3.2 Putting it together

After we have described the individual components of the EMODE tool-chain, we now focus on the following:

- integration/ interaction of individual components
- usage of the tools according to the EMODE methodology

3.2.1 Interaction concepts

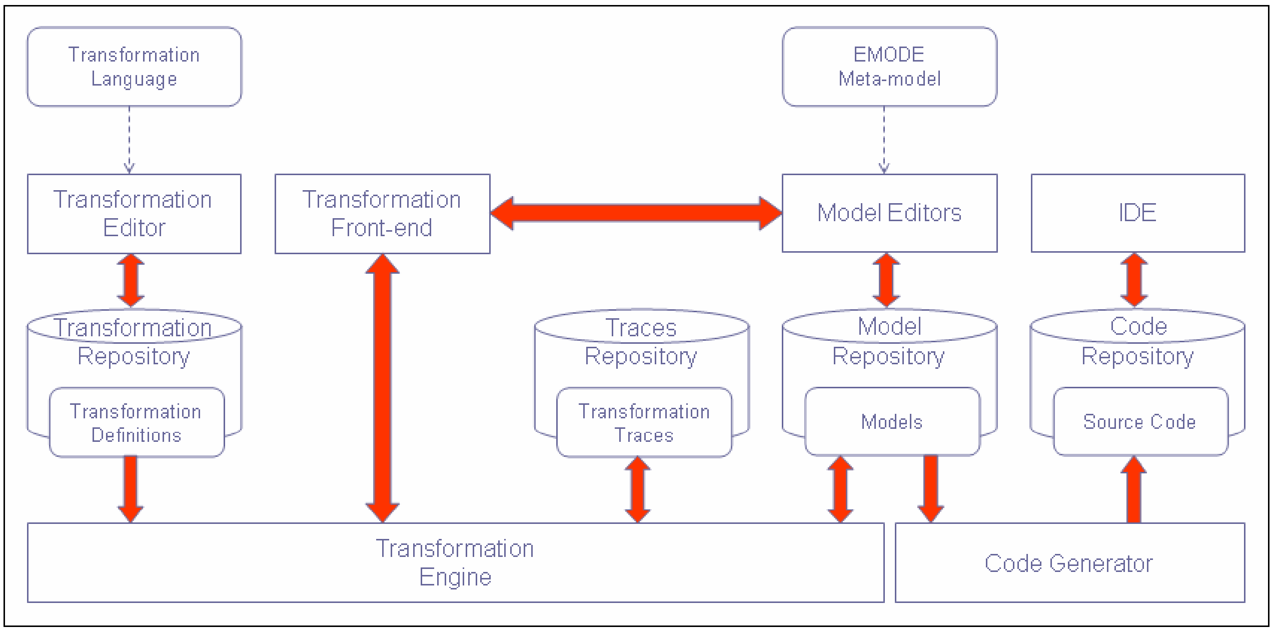

Figure 7 Interaction between tool-chain components

As illustrated in Figure 7, there is a direct interaction between the following components:

Model editors and model repository
There is a direct connection between the model editors and the model repository. Every editing step is mapped onto an API call to the repository. This way, developers working on the same repository will “see” changes of the other developers as they happen. Besides this, the model editors also allow reading and writing of models from and to XMI files, in order to exchange models with external developers not using the EMODE tools.

Model editors and transformation front-end
The transformation front-end will be integrated into the model editors, possibly as a plug-in to the model editors. This way, the developer will be able to initiate transformations directly out of the editor holding the source model.

Transformation editor and transformation definition repository
The transformation editor will be connected to the transformation definition repository the same way as the model editors are connected to the model repository. Similar to the model editors, the transformation editor will also allow the import of transformation definitions from files.

Transformation engine and transformation definition repository
The transformation engine will be connected to the transformation definition repository in order to read the transformation definition to be executed. This interaction will happen on API-level.

Transformation engine and transformation traces repository
The transformation engine will write transformation traces into the corresponding repository. This interaction will also happen on API-level.

Transformation engine and model repository
The transformation engine will have read/ write access the model repository. Again, this will be on the API-level.

Transformation engine and transformation front-end
The transformation engine will further exchange data with the transformation front-end. The latter is used to control the transformation engine and display its output to the developer. This interaction will most likely happen at API-level.

Code generator and model repository
Here the same applies as for the transformation engine.

Code generator and code repository
The code generator will be connected to the code repository by the repository-specific interaction protocol. For example, for SVN this could be HTTPS.

IDE and code repository
The IDE will communicate with the code repository in the same way as the code generator.

3.2.2 Using the EMODE tool-chain

In the following we give a brief description of how to use the proposed tool-chain. The standard workflow is illustrated in Figure 8.

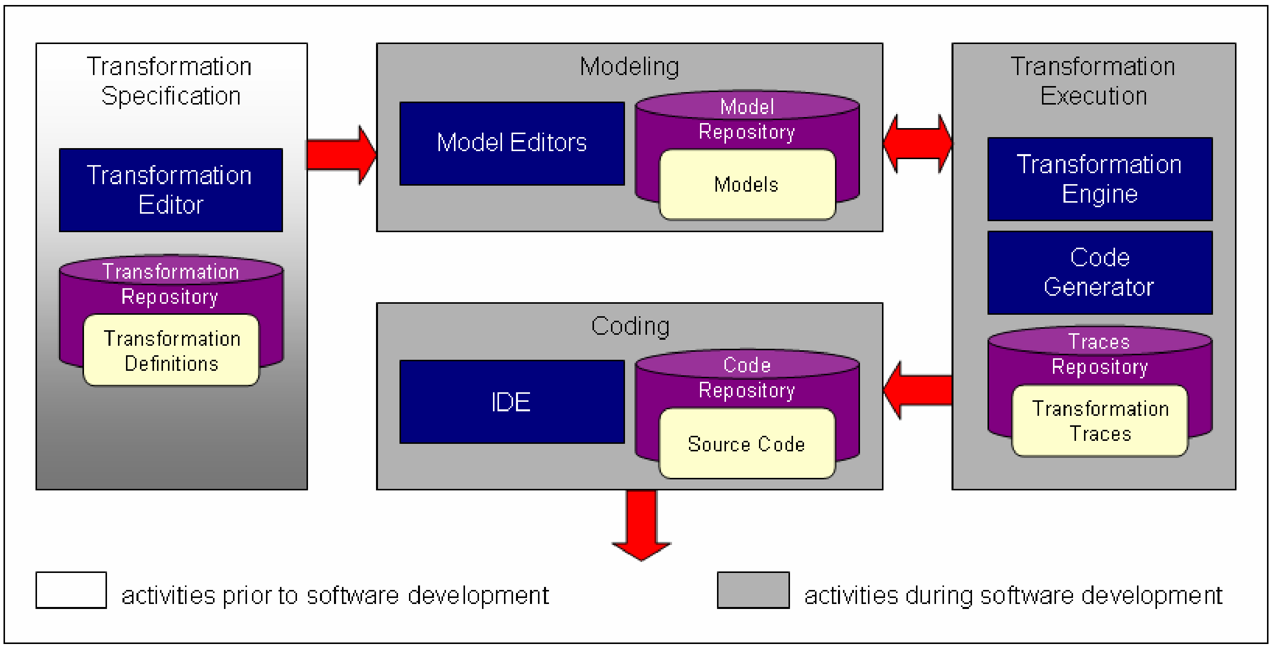

Figure 8 Using the EMODE tool-chain

Using the transformation editor, transformation definitions based on the EMODE meta-model are defined. Because the transformation definitions rely only on the meta-model itself, they can be written in advance and then be used by a number of projects. The task of writing transformation definitions should be accomplished by a specialist, because it requires comprehensive knowledge about the transformation language and tools.

During a concrete project, the application under development is being described thoroughly using models. These models are kept in the model repository.

During the modeling process, the developer can now use the transformation engine to execute the transformation definitions specified before. Cases where the developer could use transformations include, but are not limited to:

- re-factoring models
- adding more details to models
- mapping the application onto a specific technology

The transformation engine accesses the model repository in order to query source models and generate target models. In case of incremental transformations, it also accesses the traces repository to identify the elements which have already been generated. The engine executes the transformation rules contained in the transformation definition provided by the developer and generates transformation traces along with the target model element. These traces are written to the traces repository.

Prior to the actual transformation process the developer needs to set up the transformation using the transformation front-end. After the transformation has finished, the results will be displayed using the same tool.

Once the models include sufficient detail they can be transformed into executable code using the code generator. The generated code is written to the code repository. There it can be accessed, modified, and compiled using the IDE. In some cases, this last step can be omitted. This is the case if the models already contain enough information so that they can be directly transformed into machine code of the target platform.

3.2.3 Development in EMODE

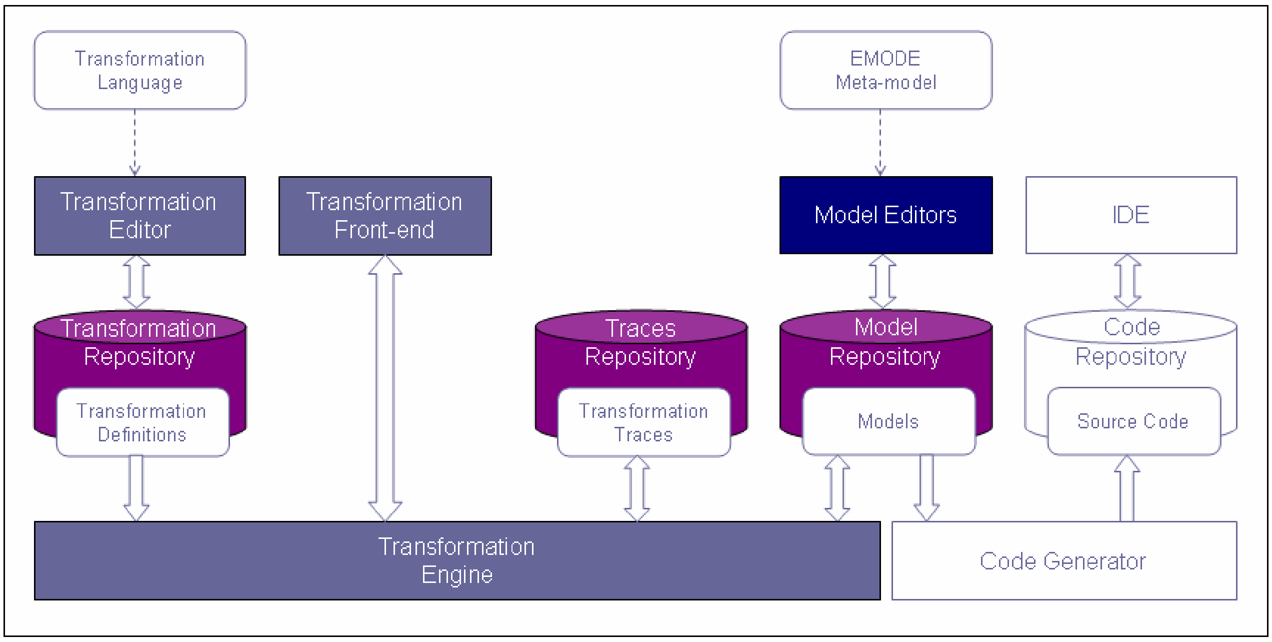

Figure 9 Tools that are to be developed in EMODE

Figure 9 shows the components of the tool-chain that need to be developed in the course of the EMODE project. They include:

- the **model editors** – they will be developed from scratch based on Eclipse [9]
- the **transformation editor** – initially, a plain text editor will do
- the **transformation engine** – this will be the QVT engine of IKV++
- the **transformation front-end** – this will go into the model editors
- the **repository for transformation definitions**
- the **repository for transformation traces**
- the **model repository** – generated from the EMODE meta-model using IKV++ medini [5]

Tools that already exist and therefore will not be developed in EMODE include:

- the **transformation definition language** – here we use QVT relations
- the **coding tools** – here we use an existing IDE such as Eclipse
- the **code repository** – here we use an existing versioning system such as SVN

4 References

1. The Object Management Group, MDA Guide Version 1.0.1. 12 June 2003, http://www.omg.org/docs/omg/03-06-01.pdf
2. K. Czarnecki, S. Helsen, Classification of Model Transformation Approaches. In *Workshop on Generative Techniques in the Context of Model-Driven Architecture, associated with the 18th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, October 2003, Anaheim, California, USA
3. A. Kleppe, J. Warmer, W. Bast, MDA Explained, The Model Driven Architecture: Practice and Promise. Addison Wesley, 2003
4. J.-M. Jézéquel, Model Transformation Techniques, last visited June 21st, 2006, http://modelware.inria.fr/static_pages/slides/ModelTransfo.pdf
5. IKV++ Medini Meta Modeler, last visited July 21st, 2006, http://www.ikv.de/index_en.htm
6. World Wide Web Consortium, XSL Transformations (XSLT) Version 1.0, W3C Recommendation, 16 November 1999, http://www.w3c.org/TR/xslt
7. The Object Management Group, MOF QVT Final Adopted Specification, 01 November 2005, http://www.omg.org/cgi-bin/apps/doc?ptc/05-11-01.pdf
8. Kermeta Project, last visited June 21st, 2006, http://www.kermeta.org/
9. Eclipse Platform, last visited July 21st, 2006, http://www.eclipse.org