EMODE

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

EMODE Deliverable D2.4: Version 2 of Metamodel and Transformations

Telecooperation Report No. 8,
The Technical Reports Series of the Telecooperation Research Division,
TU Darmstadt
ISSN 1864-0516

Written by (in alphabetical order):
Alexander Behring, TU Darmstadt
Andreas Petter, TU Darmstadt
Rene Neumerkel, TU Dresden

DOCUMENT INFORMATION	
TYPE	Deliverable
ID	D2.4 Spezifikation Version 2 (Meta-Modell / Modelltransformationen)
DUE DATE	September 30th 2007
WORK PACKAGE	WP 2. Models and Methods
PROJECT	**01ISE02 EMODE** **Enabling Model Transformation-Based Cost Efficient Adaptive Multi-modal User Interfaces**

DOCUMENT STATUS		
ACTION	**BY**	**DATE (dd.mm.yyyy)**
SUBMITTED	TU Darmstadt	24.10.2007
WP LEADER	SAP	
APPROVED	IKV++, SAP	24.10.2007

REVISION HISTORY			
DATE (dd.mm.yyyy)	VERSION	AUTHOR	COMMENT
05.09.2007	0.1	TU Darmstadt	Initial Outline
22.09.2007	0.1	TU Dresden	Input for M2C integrated
25.09.2007	0.2	TU Darmstadt	Feedback from partners integrated
28.09.2007	0.25	TU Darmstadt	Feedback from partners integrated
24.10.2007	1	TU Darmstadt	Feedback from partners integrated

AUTHORS' CONTACT INFORMATION				
NAME	ORGANISATION	EMAIL	TEL	FAX
Alexander Behring	TU Darmstadt	behring@tk.informatik.tu-darmstadt.de	06151 / 16 - 6670	- 3052
Andreas Petter	TU Darmstadt	a_petter@tk.informatik.tu-darmstadt.de	06151 / 16 - 6670	- 3052
Rene Neumerkel	TU Dresden	neumerkel@rn.inf.tu-dresden.de	0351 / 4633-8380	- 8251

Table of Contents

1 Introduction

Deliverables D2.2 [Emode 2006a] and D2.3 [Emode 2006b] presented the first versions of the EMODE metamodel, respectively transformations used in EMODE. After delivering the first versions, constant feedback was collected from tool developers (work package 3) and demonstrator developers (work package 4) in order to improve metamodel and transformations. The integration of feedback and continuous development was done in tight cooperation between teams developing metamodel, transformations (model to model as well as model to code), tools and runtime development, because the elements are tightly interconnected and build upon each other, as depicted in Figure 1.

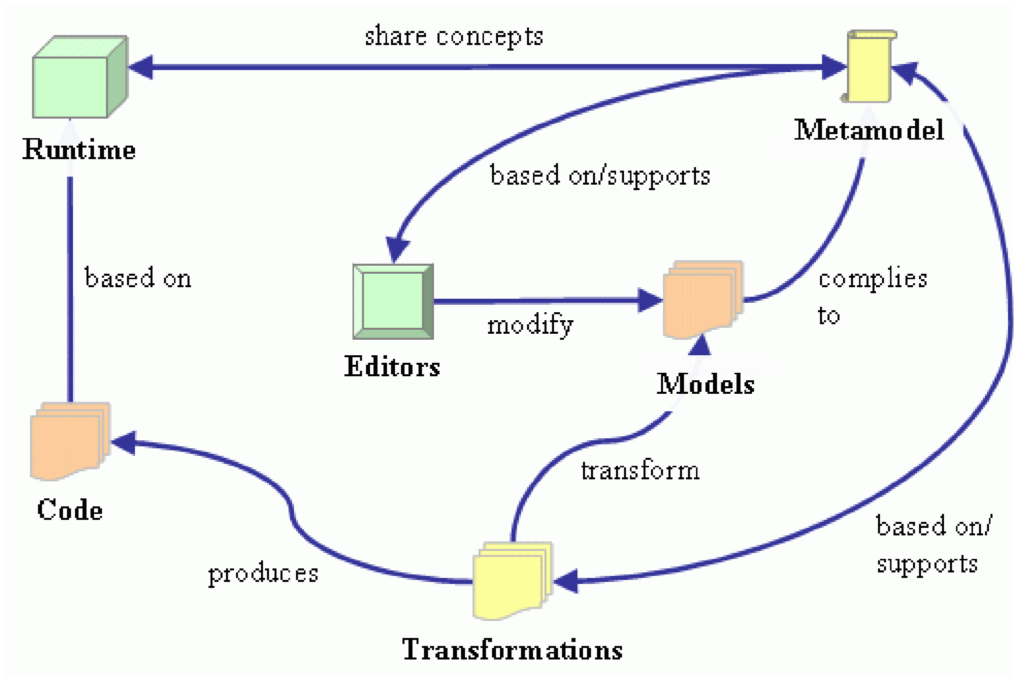


Figure 1: Illustration of the main (mutual) dependencies between the EMODE metamodel and other artifacts of the EMODE project as seen from the metamodel point of view. Yellow depicts standardized artifacts, defined by the EMODE project. Green depicts software components build by the EMODE project. And red illustrates artifacts produced and/or modified by the developer of the (e.g.) demonstrator application.

Rollout of changes mostly went from “top to bottom”. A new metamodel was released. From this, a new repository was generated to host the models compliant to the new metamodel. The transformation engine was then integrated into the repository. Model-to-model transformations were modified to reflect the changes in the metamodel. Parallel to this, the runtime was altered to support the newly introduced concepts. Editors and model-to-code transformations were adapted to support the changes made, and finally, the new revision was tested.

Feedback and ideas for improvements hereby came from all partners. As every partner had a different responsibility (transformations, tools, …), this helped to explore the concepts used and their definitions from different points of view. Beyond this, test prototypes were built, especially before releasing a new version of the tools to the demonstrator developer teams to ensure the quality of the produced tools.

In the following, this document gives an overview of version two of the metamodel and transformations, after major changes and revisions have been made. Section 2 hereby is devoted to the metamodel, and section 3 to the transformations developed in EMODE.

2 Metamodel

Like a schema definition for XML, a metamodel defines what a valid model is. It provides the abstract syntax of the defined model, i.e. it defines the syntax in a notation independent way ([OMG 2007]). Thereby, it specifies the concepts that can be used to build models that are compliant to the metamodel.

EMODE utilizes MOF 1.4 (cf. [OMG 2006]), a standard that metamodels are expressed in, in order to facilitate improved reuse and standardization of metamodels. The Enterprise Architect (EA) UML tool[1] was used to specify the metamodel. A plug-in to EA, the tool medini meta modeler[2] by the partner IKV++ enabled EA to built MOF 1.4 compliant metamodels. Medini was also used to generate the repository for model storage. This repository allows access from clients written in various programming language via its standardized CORBA interface. In EMODE, all clients have been written in Java.

2.1.1 Brief Overview

In the following, only a brief overview over the structure of the metamodel is given. Section 7 (in the separate document "EMODE Deliverable D2.4: Appendix D – Metamodel Documentation") contains the comprehensive documentation of the metamodel created while building it.

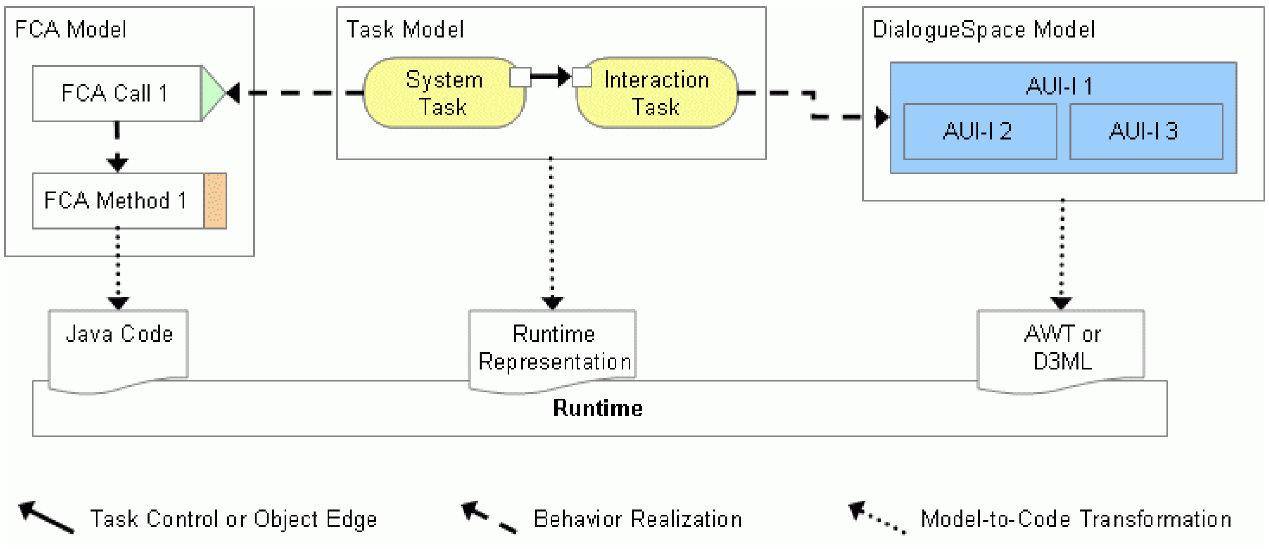


Figure 2: Sketch of the main models and their elements with inter-package associations. AUI-I stands for AUI Interactor.

In Figure 2, models built with the main packages FCA, Task and DialogueSpace are sketched. The Task package is the central package. Its model elements are used to describe behavior of application, user and interaction. The more specific description of interaction and application logic is given with the DialogueSpace and Functional Core Adapter (FCA) packages. These main packages reflect the functional decomposition made in the Arch Model [UIMS-WS 1992], except that the two presentation parts of the Arch Model are represented by a single package - the DialogueSpace.

[1] Enterprise Architect is a UML modeling tool. It can be found at http://www.sparxsystems.com.au/products/ea.html.
[2] For more information, see http://www.ikv.de/index.php?option=com_content&task=view&id=17&Itemid=31

Further packages in the metamodel are omitted for brevity from the figure. The DomainConcept package is a base providing the relevant application specific concepts. The Modality package provides a way (via key-value pairs) to formulate requirements on the interaction modalities, for designing a UI for a specific situation. Context providers and consumption are described in the Context package, the Goals package is used to model development goals, and the Commons package introduces common concepts for all packages.

2.2 Major Changes since Version One

In this section, a brief and by no means complete overview over the changes since the deliverable 2.2 (cf. [Emode 2006a]) is given. 6 contains the change log written while modifying the metamodel.

Among the changes were:

- *Support for Cursor Control*
  Used by the Multimodal Service Component in order to determine which element to activate next, e.g. for a voice modality.

- *Removed Structured Task Execution node*
  As a Task Definition node (the class notion for tasks) can be used to capsule subtasks, too, a separate element for task model structuring seems dispensable.

- *Support for AUI Refinement*
  Elements and associations to support AUI Refinement integrated.

- *Support for asynchronous communication and eventing defined*
  Abstract Event Providers and Event Consumers were defined, as well as concrete derivations of these. One derivation is a Context Event Provider, the integration of the context service into the models.
  Asynchronous communication is supported by pins that can be used without ending a task. Furthermore, semantics (e.g. to close a task) can be attached to pins.

- *Abstract Task Nodes added*
  Besides User, System and Interaction Tasks, now Abstract Tasks can be modeled. They make explicit (as in CTT, cf. [Paternò 1997]) the structuring of tasks into more abstract tasks.

- *Tool support improved / added*
  Besides a separate package to support traces for the transformation engine, a Diagramming package supports making persistent the visualization of models.

- *Removal of UML elements*
  In order to reduce the number of elements in the metamodel and clean up unused elements, UML was removed. Hereby, the notion of a class and instance was introduced and is supported by the DialogueSpace model.

- *Integration of OWL / RDF*
  The Domain Concept package now supports RDF and OWL models. They are used to model the domain of the application. OWL and RDF concepts were integrated into the

EMODE metamodel.

- *Concept Value Access*
  In order to make explicit the data exchange between User Interface and tasks, the notion of a “Concept Value Access” (CVA) was introduced. With a CVA, an element can access another element that carries a value in order to read or write the value.

- *Unused elements were removed*

2.3 Supported Key Features

The developed EMODE metamodel supports a set of key features that are used in the EMODE project. This section enumerates and describes them.

2.3.1 Modeling of Multimodal UIs Delivered Through the EMODE Runtime

The runtime delivering the multimodal UIs is described in [Emode 2006c]. Modeling these UIs is supported by the EMODE metamodel. A UI is captured in an AUI Box element. I.e. inside an AUI Box, the two main elements that can populate an User Interface are arranged:

- **AUI Interactors** – an interactor is an UI element supporting a defined task (like a combo-box that supports a selection of an item from a list). It can be given a type (e.g. the combo box or a data field) that defines what task it supports. Note that interactors can be nested and contain zero or more other interactors. And
- **AUI Space** – an AUI Space is an element that serves as a place holder to be filled at runtime with an interactor. For example a wizard is built by placing the interactors for the different pages into a space – one after the other.

AUI Interactors and AUI Spaces are arranged inside an AUI Box to describe an User Interface.

In contrast to other approaches (e.g. [Limbourg 2004, Mori 2005]), where the interactor types are hardcoded into the metamodel, interactors can dynamically be typed in EMODE. A class-instance relationship for interactors allows defining the possible types in an external library. It is still formal, as the library is made up of AUI Interactor Classifier elements. This way, no fixed level of abstraction is introduced. The level of abstraction of an UI (AUI Box) is given by the level of abstraction of the interactor types used in its definition. This approach also keeps the metamodel small and is much more flexible for integrating new interactors and modalities. E.g., EMODE integrated a Google Map[3] interactor upon request by developers – no change of the metamodel was necessary.

The building of a User Interface is supported by the AUI Refinement process, a technique created in EMODE that leverages MDA [Miller 2003] for the development of User Interfaces. It is explained in the following section.

2.3.2 AUI Refinement

MDA is based on the concepts of models at different levels of abstraction and transformations between them. In MDA, abstract models are hereby transformed into more concrete ones. If the later is interpreted as an abstract model, the transformation into an (even more) concrete model can be executed once more.

We call the context that a UI model is refined for a “situation”. Situations can be differing in device, platform and/or other context parameters. More abstract situations (e.g., “meeting”) can be

[3] Google Maps, a popular GIS service, can be utilized via an API (http://www.google.com/apis/maps/).

specialized into more concrete ones (e.g., “meeting with colleagues” and “meeting with soccer team”).

When identifying the UI models (designed for different situations) with nodes in a graph, the edges in this graph reflect the specialization relationships between the situations. Hence, the graph is directed and acyclic, with more abstract models sorted before more concrete ones[4]. I.e. the more abstract models can be found closer to the root of the graph (cf. Figure 3). To pick up the meeting example from above, there is a more abstract model for the situation “meeting” and two more concrete models for the two more concrete meeting situations. The two more concrete models are connected via directed edges to the more abstract model (“meeting”).

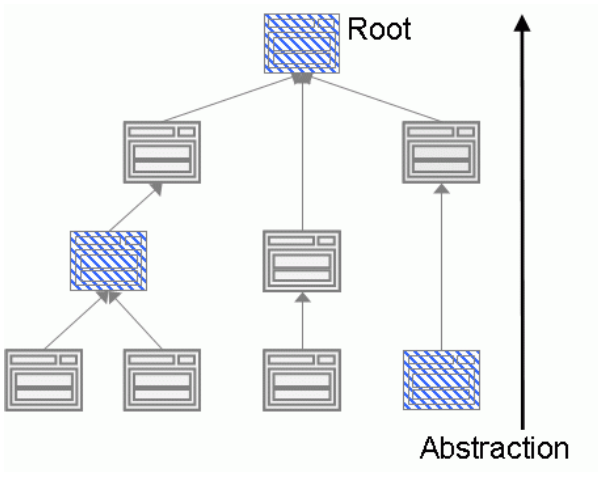


Figure 3: Sketch of an AUI Refinement Graph with AUI Boxes and their contents.

A node in the graph is called a refinement and is in the metamodel represented by a box element (called AUI Box). In this box element, all model elements specifying the application’s UI for a given situation are captured. Hence, every node of the graph represents an UI of the application, with the most abstract UI model being represented by the root node. The graph formed by the AUI Boxes (as depicted in 1) is called AUI Refinement Graph.

If an AUI Box is refining another AUI Box (i.e. it is a child of it in the AUI Refinement Graph), it must be as or more concrete as the box it refines. Thus, the situation that the more concrete (child) AUI Box is used for must be contained in the situation that the more abstract (parent) AUI Box is used for.

During development the AUI Refinement Graph is built by producing UI models at different levels of abstraction. The most concrete model then either is transformed into code (as in [Limbourg 2004, Mori 2005]) or interpreted at runtime.

2.3.3 Integration of the Modeled UIs with the Application Logic

The approach to integration of User Interface and application logic is shown in Figure 2**Fehler! Verweisquelle konnte nicht gefunden werden.**. The task model serves as a central connection point. Its data and control dependencies between tasks coordinate and mediate information between system and interaction tasks. These in turn are mapped onto FCA Methods (that represent the application logic) and AUI elements respectively.

[4] I.e. the models in the tree are topologically sorted.

2.3.4 Metamodel with Comprehensive Support for Building EMODE Applications

Most of the application code can be generated by model 2 code transformations, except application (business) logic elements (cf. section 3 for more details). This also includes generation of the dialogue controller, application logic stubs and connections to the context server.

2.3.5 Support of the EMODE Tool Chain

Besides the actual description of the application, tools need to store information with the models. Graphical editors require layout information and the transformation engine requires traces to keep track of previously executed transformations. To avoid producing a variety of files, all referring to the same model, the EMODE metamodel allows storing this tool-specific information with the application description in the model.

2.3.6 Definition and Consistent Combination of a Set of Concepts Needed to Develop Multimodal, Context-Aware Applications

All features listed in the previous sections are built into the metamodel. First prototypes built show that the set of defined concepts is sufficient to develop multimodal, context-aware applications. No inconsistencies in the metamodel were discovered through the prototypes and model-to-code transformations. Associations are used to connect aspects related to each other (e.g. the UI elements and Interaction Tasks as in section 2.3.3 described). Transformations (as a form of a mapping) were introduced between different models (cf. section 3.1 on model-to-model transformations).

2.4 Conclusion and Outlook

The changes presented in the last section and in section 6 were introduced in the metamodel to support new concepts and due to feedback from developers (runtime, transformation, editors and demonstrators). Ideas delivered in D2.2 were improved through the integration of the feedback.

The revision of the metamodel increased its applicability for modeling multimodal, context-aware applications. As for now, complex concepts such as the event-based integration of the task model with the context server and complex interactors such as a GoogleMap can be modeled.

The EMODE metamodel supports the key features:

- a consistent integration of key concepts to develop EMODE applications,
- modeling of multimodal User Interfaces,
- integration of UIs with the application logic,
- comprehensive support to build EMODE applications,
- AUI Refinement, and
- EMODE tool chain support.

The EMODE metamodel hereby differs to other approaches especially in:

- The way UIs are modelled: EMODE supports AUI Refinement and does not include interactor types in its metamodel. Hence, it breaks the dichotomy of UI model abstraction level being either abstract or concrete.
- The comprehensiveness: No other models or tools are needed to develop the application. The models address all components of the Arch-model [UIMS-WS 1992] and further support modelling context provision and use, as well as requirements on supported modalities.

Currently, the evaluation of EMODE is underway. We are positive that the development of the two demonstrators will be a validation that building multimodal, context-aware, adaptive applications with the metamodel produced in EMODE is a promising approach.

It is already clear that further work at the Telecooperation Group at the TU Darmstadt will be based on the metamodel developed in EMODE.

3 Transformations

This part of the document presents the improvements made in developing the EMODE transformations starting from D2.3. Starting from the definitions of model-to-model transformations given in D2.3 we describe the implementations done during the project in this document.

The transformations used in model driven engineering may be partitioned into two categories [Czarnecki 2003]:

1. Model-to-Model Transformations, which transform models into models and
2. Model-to-Code Transformations, which transform models into source code.

While the first category mainly targets automation of the modeling process and provides support for developers, the second category aims at deploying the application and bringing it into a form that can be modified in more detail. Normally model-to-code transformations transform models into source code. However, due to the use of the EMODE runtime environment the model-to-code transformation used in EMODE transforms the models into a mixture of XML representation of the models and source code.
The first part of the transformation chapter introduces model-to-model transformations used in EMODE, while the second one introduces model-to-code transformations.

3.1 Model-to-model transformations

According to the MDA Guide [Miller 2003], a *Model Transformation* is *“the process of converting one model to another model of the same system”*. They have a wide range of applications within the EMODE project. The two main purposes are:

- to be executed in order to transform one model into another one
- to ensure consistency between two models that have mappings between them (relational/multidirectional).

With *Transformations*, we denote all transformations that occur during the design process of the application (see e.g. [Czarnecki 2003] for a definition of terms). This may range from early project phases, like a high-level semi formal project description to very late phases, like building deployment packages. Hence, they are used to support the development process, which is the focus of the transformations introduced in this section.

Design Time Transformations can occur as model-to-model or model-to-code. Theses two types are introduced in the following subsections.

3.1.1 Model-To-Model Transformations

Model-To-Model Transformations produces models out of other models. A model is defined in the MDA Guide [Miller 2003] as: *“A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language.”*

Hence, a model is a description of the system, often capturing information about a specific aspect of the system (e.g., a task model which describes tasks of the system). Consequently, models can overlap in their contained information, or one model can capture the same information as another one, only with more implementation relevant details. To synchronize the overlapping aspects and produce the already existing facts for a new model from a pre-existing one, the need for transformations arises.

3.1.1.1 Model-To-Model Transformations according to the MDA

This subsection introduces some of the concepts advocated in the MDA Guide [Miller 2003]. It briefly describes what types of Design Time Transformations exist. Hereby it must be noted that all transformations introduced in the MDA Guide have generative character and require a Platform Independent Model (PIM), which is transformed into a Platform Specific Model (PSM) utilizing a Platform Model (PM).

$$PIM \xrightarrow{PM} PSM$$

The PM is left out in the following, since in all cases, it is either implicitly included within the source model or hard-wired within the transformation source code.
A transformation can be in-place, which means that source and target models refer to the same model instance. This type of model-to-model transformations is used extensively in EMODE, as noted below.

Generative Transformations
Generative Transformations have one or more target models that they operate on. The source models deliver input information which (by the transformation rules) specifies how the target model elements should look like. Hence, the transformation generates the elements in the target models.
The major advantage of using transformations is based on (partial) automation of the modelling process. Model elements generated by transformations do not need to be modelled by hand.
Model-to-Model-Transformations can be applied in the following cases:

- Models can be derived automatically (to some extent)
- Transformations can modify information in the model automatically (to some extent)
- When a source model has been changed by the developer, the target models can be modified using the transformations, such that change propagation does not need to be done by hand (to some extent).

Relational Transformation
In Relational Transformations, domains may be used to specify sets of model elements from different or equal models. These domains may be annotated by read and write modifiers (In case of QVT Relations these are called “checkonly, enforce”) and a set of expressions to handle attribute values. A transformation (or a relation between domains) may then be defined by giving several domains. If several domains are specified to be writable, the transformation may be used as a relational transformation. The developer has specified a relation between the domains. Consequently, it can be used to ensure consistency between the models, that is, if the configuration in question will allow for that, the transformation can be applied in both directions, switching source and target models. This may be done by triggering the transformation, when consistency should be established, or automatically, every time the model is changed.

The Relational Transformations can be seen as a generalization of Generative Transformations. If a Relational Transformation is defined and its source models are protected from modification, it is restricted to a Generative Transformation. Therefore, Generative Transformations can be written in a Relational Transformations language.

It is also noteworthy that Relational Transformations can very conveniently be expressed in a declarative language. The transformation engine then has to take the appropriate steps to ensure the consistency (constraint) between the source and target models, as defined in the transformation.

The language QVT Relations is able to handle relational transformations. However, it should be noted that relational transformations have not been used in EMODE, because the transformation do not allow for bi-directional application, which is required by the definition of the term relational transformation. In EMODE, Relational Transformations are specialized to Generative Transformations by restricting the direction in which they may be applied.

3.1.2 Model-to-Model-Transformations in EMODE

Model-to-model transformations increase the efficiency of application modeling based on EMODE, compared to modeling by hand. To be most productive they are aligned at the EMODE methodology, based on several phases (D2.1), that can be executed in any order. Feedback for phases being executed before the current phase is allowed, and lead to changes in phases before the current phase, resulting in feedback loops. Therefore the EMODE methodology is an iterative design process for designing interactive multimodal systems. For model-to-model transformations the design phases “High-level-design” and “Detailed-Design” are of interest, as these are part of the modeling process.
To support the developer transformations are designed to produce and update model elements based on other elements. Since all models are used within one monolithic repository, transformations used in EMODE are in-place; that is, source and target models refer to the same model instance (the term model instance is used to describe the terminologically confusing distinction between models and a concrete model, here.). Other deliverables refer to several models, which are contained in packages of the same model instance in the repository. As seen from the repository the models (as they are called in other texts on EMODE) are organized in several packages, which are part of a big model (therefore called model instance). The time transformations save in the modeling process by hand is a reduction of redundancy. The redundancy comes from semantic similarities of metamodel elements in different packages, which correspond to each other in terms of a semantic mapping. This redundancy is reduced by writing an algorithmic description of the mapping - by a transformation description - which exploits the known redundancy of model elements in different models.
This part of the document describes the transformations that have been developed in the EMODE project.
In the first section of the model-to-model transformation part, the transformations introduced in EMODE are compared to transformations introduced in other projects. In the second section the transformations that have been designed in EMODE are presented. Finally, the appendix lists some QVT Relations code that has been developed.

3.1.3 Related Work

In this section some results are presented, which have been made in similar works on transformations in the past.

3.1.3.1 Fabio Paternò

In [Paternò 2002] Paternò presented model driven development for user interfaces using model transformations. The work focused on the generation of different user interfaces for different platforms from a task model. In [Mori 2005] the approach has been enhanced by the introduction of multimodality into the modeling and transformation process. The most important modeling aspect in the Teresa tools is the CTT [Paternò 1997] task model. The user interfaces are not considered to be models (“one model, many interfaces approach”).

Compared to EMODE the transformations in Teresa mainly focus on transformations to derive user interfaces from a task model. Furthermore, EMODE uses a standard model transformation language which simplifies the reuse of the transformations and the comparison to similar transformations.

3.1.3.2 Quentin Limbourg and Adrian Stanciulescu

Quentin Limbourg [Limbourg 2004] and Adrian Stanciulescu [Stanciulescu 2006] present a model transformation tool ("TransformiXML") specifically designed to transform user interfaces. It is based on the UsiXML[5] user interface modeling language and graph rewriting (also called graph grammars). The most important model types of UsiXML are Task, AUI (abstract user interface) and CUI (concrete user interface) models. Limbourg states, that user interface can be developed both ways – using transformations – by either starting from a task model or a CUI model.
However, both do not use a standardized transformation language, but a more theoretical language based on graph rewriting. The transformations defined are not as easily reused as if they were written in a standardized transformation language.

3.1.3.3 Jean-Sébastian Sottet

In [Sottet 2006] Sottet introduces models at runtime to sustain usability even in the case of unknown environmental situations or platforms, called ("plasticity of user interfaces"). His work is based on prior work [Demeure 2005] in which a basic approach to handle open adaptivity [Schmid 2006] is presented.
Compared to the work presented here, he uses a non-standardized model transformation language called ATL and transforms models at runtime. Though being a very good and feasible approach his work may not be easily compared to the work presented here, because transformations are designed to support user interface adaptation at runtime and do not focus on helping the developer at design time.

3.1.3.4 Murielle Florins

Murielle Florins [Florins 2006] provides an extension to Limbourg and Stanciulescu by extending their work using rules defined in OCL to define usability in the context of transformations. Usability Guidelines can be written for each type of modality, though Florins only defined GUI guidelines, in her thesis.
In her work, she focused on the AUI to CUI and CUI to FUI transformation and not – as in EMODE – on the whole process. Though guidelines help in developing user interfaces the transformation developed by Florins do not take into account existing model elements in the target model which is required by the EMODE methodology. Implementation of guidelines would have been a plus for EMODE, but has not been implemented due to timely issues.

3.1.4 Transformations defined in EMODE

The model-to-model transformations defined in EMODE are introduced in this section. They are written in QVT Relations - a standard transformation language defined by the OMG [OMG 2005]. The QVT Relations transformation engine has been implemented during the EMODE project by IKV++. However, since it is part of D3.3 it will not be described in further detail here.

3.1.4.1 Methodology

Transformations have been aligned at the EMODE methodology to provide maximum support to the developer during the application of the EMODE methodology (otherwise the transformations would probably not fit the needs of the developer in terms of the modeling process). As the EMODE methodology is an iterative design process, transformations should be able to support, if

[5] http://www.usixml.org

technically feasible:

- Identification of previously hand-modelled model elements, which have been modeled before and the prevention of model element recreation in the target model.
- When transformations are executed several times, the transformations should be able to remember which model elements have been created or modified during previous transformations by looking at a transformation trace model. The previously created model elements should be reused and should be deleted or modified.
- Under certain circumstances transformations should be able to provide forward and reverse transformation rules, if the methodology allows the developer to either edit source and target models at the same time and redundancy between model concepts is high.
- Refinement of models can be supported by transformations if the developer would have to redesign most of the elements in the refined model.

Figure 4 shows the impact of the EMODE methodology (D2.1) on the EMODE model-to-model transformations. The analysis performed by the developer is done while developing Goal, Task and DialogueSpace Models, where the developer adds goals, tasks and AUIInteractors. While doing that, he might notice that he wants to switch to a different model to continue modeling at a different model (e.g. switch from goal modeling to task modeling). He then transforms his model to the model where he wants to continue editing and switches to the right editor to make the changes.
Special cases are the DialogueSpace and FCA transformations. The DialogueSpace transformation takes into account AUI Refinement and is invoked on DialogueSpace models only. Depending on the type of transformation, transformations have been implemented in JAVA or QVT. Almost all transformations have been implemented in QVT Relations, only the DialogueSpace-to-DialogueSpace transformation has been implemented in JAVA. The FCA model can either be generated – if the Systemtasks are known at development time, first, or it can be the starting model for the process and be used to produce Systemtasks. Both transformations have been implemented in EMODE.

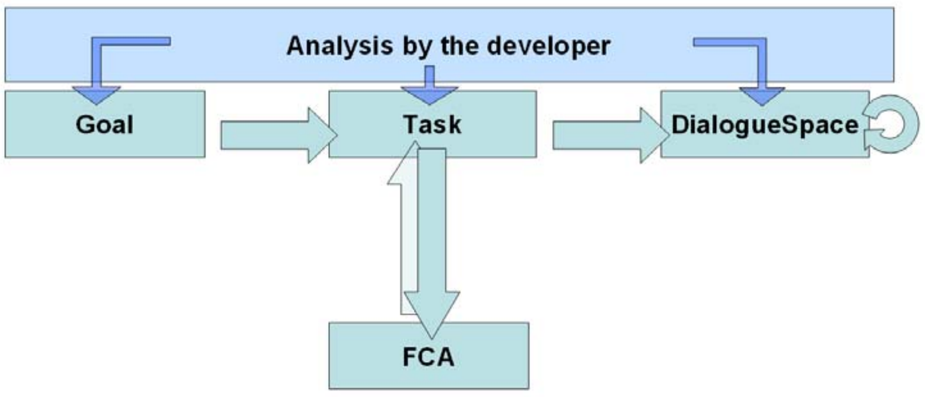


Figure 4: Transformations defined in EMODE aligned at methodology. Transformations are presented in big arrows while the smaller ones depict information provided by the developer.

3.1.4.2 Transformation Process Triggered Using EMODE Editors

The transformations may be invoked directly from the editors, which operate on the repository. This repository also includes a QVT Relations transformation engine which executes the QVT Relations transformations. For executing a transformation their definition is read from a corresponding input file and converted into a CORBA String representation. This String representation is then sent to the locally executed CORBA based model repository, to be executed by the QVT engine. The result is sent back to the editor to display error messages.

3.1.4.3 Algorithms

As seen in Figure 4, several transformations have been defined:

1. Goal-to-Task-transformation
2. Task-to-DialogueSpace-transformation
3. Task-to-FCA-transformation
4. FCA-to-Task-transformation
5. Task-to-AUIMM-transformation
6. DialogueSpace-to-DialogueSpace-transformation
   a. JAVA
   b. QVT

In the following sections the transformations are being described shortly. A simple figure provides an example. The full QVT source code of transformations can be found in the appendix.

Goal-to-Task

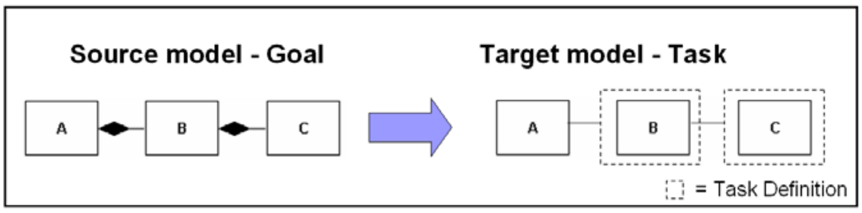


Figure 5: Goal-to-Task Example

Goal-to-Task-transformations are a good point for starting the transformations. According to the methodology, goals are the first model elements to be modeled. Goals are divided into *FunctionalGoals* and *NonFunctionalGoals*. Only *FunctionalGoals* are transformed into *AbstractTasks*. As goals may be structured into a tree, the structure is copied over into the task model by providing corresponding task definitions for all goals which are not leaf nodes of the tree of goals.
In Figure 5 an example is given. *Goal* A is transformed into *Task* A, while *Sub-Goals* B and C are transformed into *Sub-Tasks* B and C with corresponding *Task definitions*.

Task–to-DialogueSpace

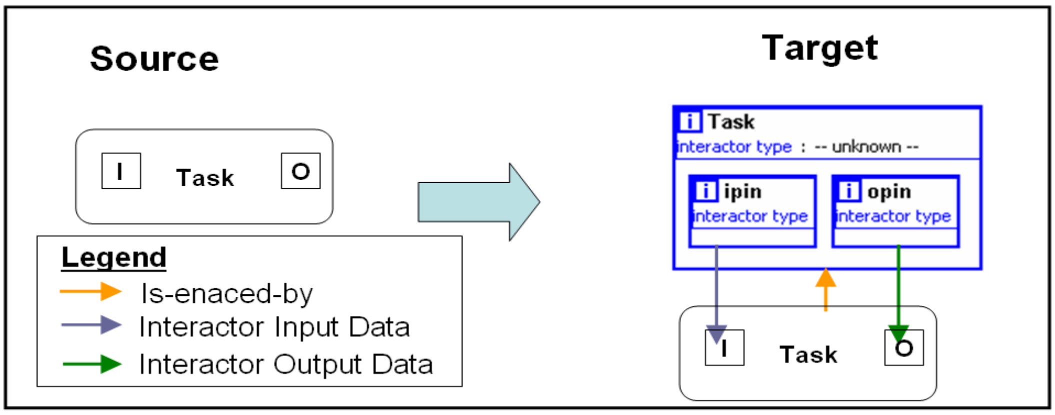


Figure 6: Task-to-DialogueSpace Example

The Task-to-AUI-transformation transforms Interaction Tasks to AUIInteractors with their URL being set to match the URL of a container (because they are often used as container components). In- and OutputPins are transformed into Child-interactors of the AUIInteractor.
Task Definitions are transformed into standalone AUIInteractors which are presented by the runtime by inserting them into placeholders (AUI Spaces). Figure 6 gives an example: Task with name "Task" is transformed into AUIInteractor "Task". The pins "ipin" and "opin" of the task are transformed into AUIInteractors "ipin" and "opin" being children of the AUIInteractor "Task".

Task-to-FCA

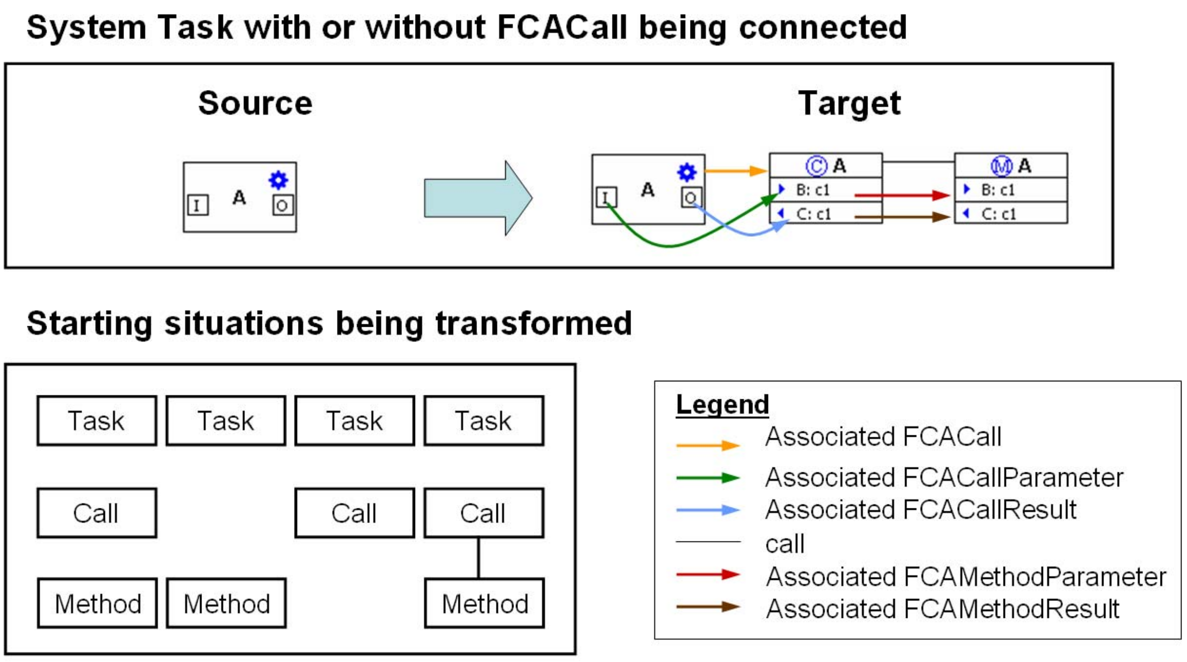

System Task with or without FCACall being connected
Source
Target
I
A
O
I
A
O
A
B: c1
C: c1
A
B: c1
C: c1
Starting situations being transformed
Task
Task
Task
Task
Call
Call
Call
Method
Method
Method
Legend
Associated FCACall
Associated FCACallParameter
Associated FCACallResult
call
Associated FCAMethodParameter
Associated FCAMethodResult

Figure 7: Task-to-FCAExample

The Task-to-FCA transformation aids the developer in the process of developing system methods or system functions by providing necessary model elements automatically. Every *System task* is transformed into a *FCACall*. *FCACalls* are then transformed into *FCAMethods*. *In-* and *OutputPins* are transformed into parameters and results respectively. Concepts are copied. Figure

7 provides a short example. *SystemTask* A with *InputPin* C (Concept c1) and *OutputPin* B (Concept c1) is transformed into *FCACall* A with *FCACallParameter* B and *FCACallResult* C. The Call is transformed into *FCAMethod* A with *FCAMethodParameter* A and *FCAMethodResult* B. *Concept* c1 is being attached to all parameters and results. The starting situations denote the situations that may have been modeled by the developer before the transformation is executed and which are handled correctly by the transformations.

FCA-to-Task

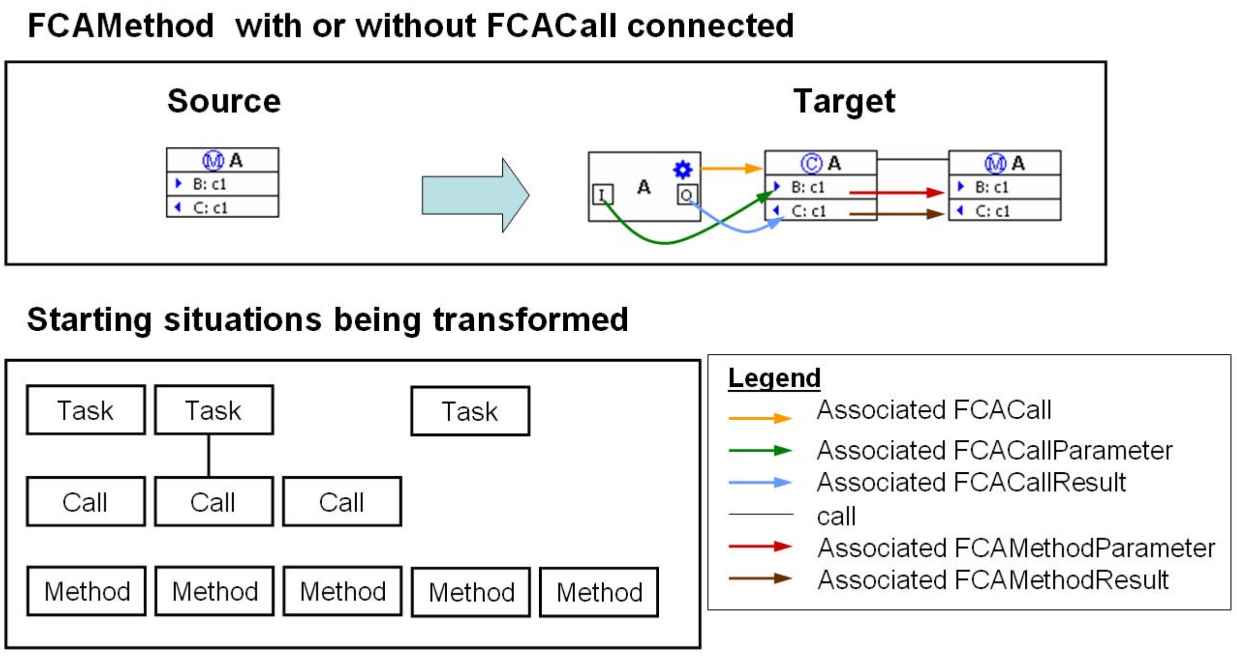

FCAMethod with or without FCACall connected
Source
Target
A
B: c1
C: c1
A
I
Q
A
B: c1
C: c1
A
B: c1
C: c1
Starting situations being transformed
Task
Task
Task
Call
Call
Call
Method
Method
Method
Method
Method
Legend
Associated FCACall
Associated FCACallParameter
Associated FCACallResult
call
Associated FCAMethodParameter
Associated FCAMethodResult

Figure 8: FCA-to-Task Example

The FCA to Task transformation may be used to transform large amounts of already available application logic encapsulated in business methods into corresponding model elements (e.g. an enterprise resource planning system might have such methods). The developer models the corresponding *methods* and transforms them into a set of *FCACalls* and *tasks*. Arguments used in the *Method parameters* are transformed into *pins* using the same *concepts* as the types of the *parameters*.
Figure 8 gives an example similar to the example given in Figure 7, except the direction is reversed, so one can state that the FCA2Task transformation may be seen as the reverse engineering process to the Task2FCA process.

Task to AUIMM

The Task to AUI transformation using Modality Information is very similar to the Task-to-DialogueSpace transformation (Figure 6). It differs by producing the same output for each ModalityRequirementsProfile defined in the modality model in addition to the one produced by the task to dialoguespace transformation.

3.1.4.4 Learnings from Developing the Model-to-Model Transformations

Several problems have been found during the development of the transformations. These are summarized in the following section.

- The semantics of the delete operation have been implemented several times in parallel. Delete Semantics have been implemented in the editors and the repository, because transformations may delete elements. This could have been prevented, if, on the one hand the editors would have called the delete operation implemented in the repository. On the other hand, if the transformations would have had direct access to the delete operations of the editors. This would require a notification system, which notifies the editor about certain events occurring during the transformation process.
- Transformations can modify, create or delete model elements. In EMODE model editors have been developed, which trigger the transformation process. However, since the transformation process is executed via CORBA, the editors have not been notified about changes in the model. Notification of the editors about the change, creation and deletion of elements would have been beneficial, considering the data needed for the graphical representation of model elements. Currently, the editors do not have the possibility to detect which elements changed during the transformation process. In the case of the EMODE project transformations were difficult to connect to the editors without integrating CORBA services into the editors. Therefore this has not been done, and the editors had to infer the state of the model before and after the transformations.
- Key elements are used to select and match elements from source and target models, such that the elements do not need to be generated using transformations (then, traces do exist, and the transformation engine can detect model elements which have been generated in a former transformation step), but can be modeled by hand in advance of any transformation step. In QVT keys for model elements can only be defined for whole transformations and only one key per model element. This raises a problem when different keys for one element are needed in a transformation. This is the case for meta-models which employ the target elements for different semantic purposes like structuring of model elements and other associations to different model elements, used in different realations.

3.1.5 Partner Feedback

The feedback regarding model-to-model transformations, as included in Appendix E: Partner Feedback, is summarized in the following. The feedback recognizes enormous potential that lies in using transformations. It then lists factors that amplified the inherent complexity of writing mode-to-model transformations. These factors can be attributed to the nature of the project (research and not product development) and the fact that QVT is in an early phase of its life as a language and therefore the engine is relatively new.

3.1.6 Conclusion

In EMODE it has been shown that model-to-model-transformation to support model driven development of user interfaces can be implemented using QVT Relations.
In addition to the evaluation of a standardized transformation language, we were able to evaluate the use of traces for model transformations, which keeps information about model elements which have been transformed. This – in combination with the key concept of QVT Relations and a set of special OCL queries to track model elements already modeled – has added to the value of the transformations in the context of iterative transformations.

3.2 Model-to-Code Transformations

As the final step in the transformation process, code generation turns design time models into executable source code. At this point, the models already contain all the information necessary to describe the application's user interface, data types, and workflow. These parts are therefore fully generated during transformation. What is still missing is the actual business logic, since this part of the application cannot be automatically generated due to its diversity. Thus, the developer will have to manually implement some of the application's logic after successful code generation. We give more detailed information when we discuss the actual transformations and their associated artifacts.

3.2.1 Overview

During the transformation process several runtime artifacts are generated from the design time models. There are different types of runtime artifacts as shown in Figure 9. Most parts of the application are specified in Java. Note that the EMODE runtime environment itself is also based on the Java platform. For the description of multimodal user interfaces, however, we use D3ML[6] as explained in [Emode 2006c]. In addition to this, some of the configuration files are defined using XML.

For the reasons given in [Emode 2006b] we use JET[7] for the generation of Java classes, whereas for the generation of D3ML we use XTL[8] instead. In both cases the transformation is a two-step process. The first step consists of extracting all the necessary information from the model repository and storing it to an easy to process data format. In the second step this data is used within the templates to produce meaningful Java or D3ML code respectively. During the transformation process, a new Java project is created in the developer's workspace, containing all the resources created during the transformation.

[6] D3ML – Device-Independent Multimodal Markup Language, developed as part of the SNOW project, visit http://www.snow-project.org for more information
[7] JET – Java Emitter Templates, a code generation facility as part of the Eclipse Modeling Framework, visit http://www.eclipse.org/modeling/emf for more information
[8] XTL – XML Template Language, currently developed by SAP, additional information is not available at this point

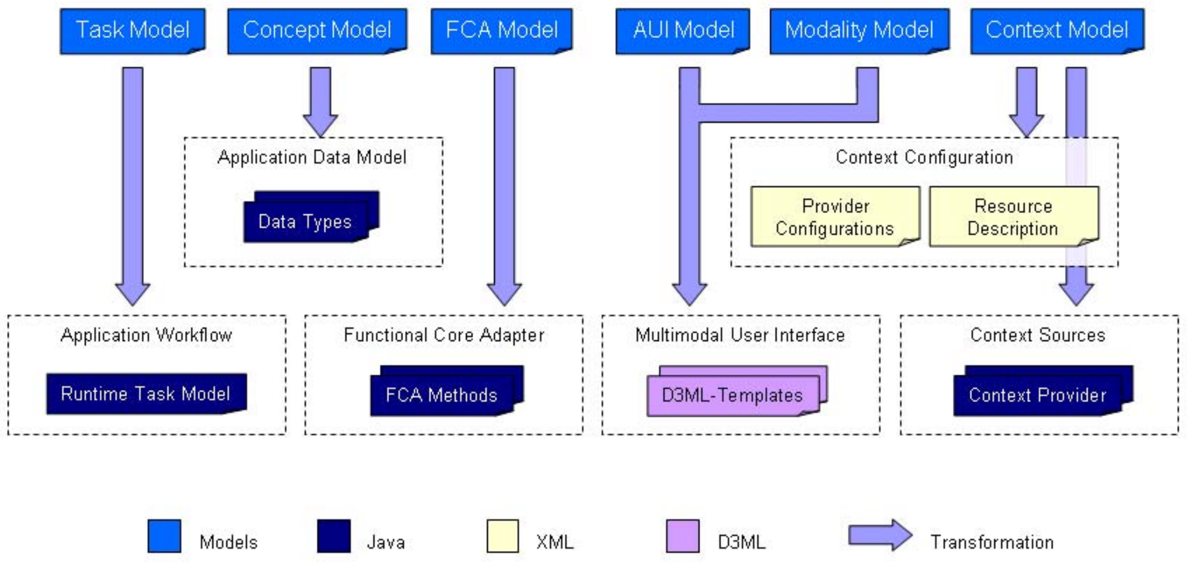
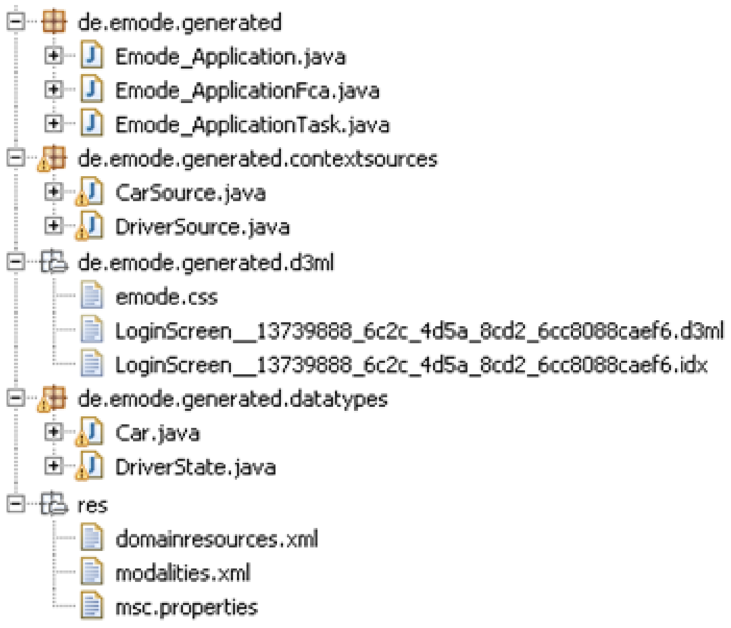


Figure 9: Transformation sources and targets

Figure 10 shows a newly created project after successful transformation. The generated resources will be explained in more detail when introducing the individual transformations.



```
de.emode.generated
    Emode_Application.java
    Emode_ApplicationFca.java
    Emode_ApplicationTask.java
de.emode.generated.contextsources
    CarSource.java
    DriverSource.java
de.emode.generated.d3ml
    emode.css
    LoginScreen__13739888_6c2c_4d5a_8cd2_6cc8088caef6.d3ml
    LoginScreen__13739888_6c2c_4d5a_8cd2_6cc8088caef6.idx
de.emode.generated.datatypes
    Car.java
    DriverState.java
res
    domainresources.xml
    modalities.xml
    msc.properties
```

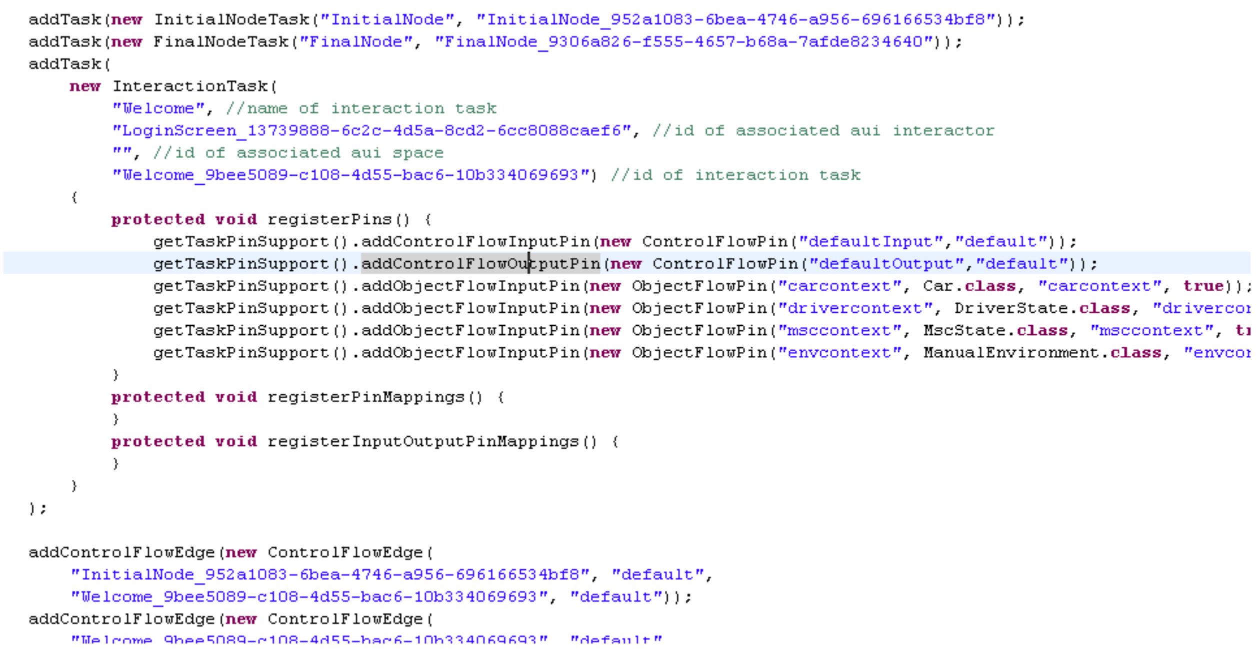
Figure 10: Example of generated Java project

In case of repeated transformation, all resources are updated to reflect the latest version of the source models. However, the source code contains especially marked sections that allow manual changes to the code which are being preserved in case of re-generation.

In the following section the different model-to-code transformations and their corresponding targets are being introduced.

3.2.2 Task2Code

This transforms the Task Model [Emode 2006a] into a Java-based representation that can be executed by the Task Process Engine. Virtually all elements in the task model are represented by corresponding Java objects comprising all the information available at design time. Figure 11 shows an example of a runtime task model.



```
addTask(new InitialNodeTask("InitialNode", "InitialNode_952a1083-6bea-4746-a956-696166534bf8"));
addTask(new FinalNodeTask("FinalNode", "FinalNode_9306a826-f555-4657-b68a-7afde8234640"));
addTask(
    new InteractionTask(
        "Welcome", //name of interaction task
        "LoginScreen_13739888-6c2c-4d5a-8cd2-6cc8088caef6", //id of associated aui interactor
        "", //id of associated aui space
        "Welcome_9bee5089-c108-4d55-bac6-10b334069693") //id of interaction task
    {
        protected void registerPins() {
            getTaskPinSupport().addControlFlowInputPin(new ControlFlowPin("defaultInput","default"));
            getTaskPinSupport().addControlFlowOutputPin(new ControlFlowPin("defaultOutput","default"));
            getTaskPinSupport().addObjectFlowInputPin(new ObjectFlowPin("carcontext", Car.class, "carcontext", true));
            getTaskPinSupport().addObjectFlowInputPin(new ObjectFlowPin("drivercontext", DriverState.class, "driverco
            getTaskPinSupport().addObjectFlowInputPin(new ObjectFlowPin("msccontext", MscState.class, "msccontext", t
            getTaskPinSupport().addObjectFlowInputPin(new ObjectFlowPin("envcontext", ManualEnvironment.class, "envco
        }
        protected void registerPinMappings() {
        }
        protected void registerInputOutputPinMappings() {
        }
    }
);

addControlFlowEdge(new ControlFlowEdge(
    "InitialNode_952a1083-6bea-4746-a956-696166534bf8", "default",
    "Welcome_9bee5089-c108-4d55-bac6-10b334069693", "default"));
addControlFlowEdge(new ControlFlowEdge(
    "Welcome_9bee5089-c108-4d55-bac6-10b334069693", "default"
```

Figure 11: Representation of the task model at runtime (excerpt)

Generated artifacts:

- *EmodeApplication.java* – The main class for starting the application
- *EmodeApplicationTask.java* – The runtime task model

3.2.3 Concept2Code

This transforms the elements of the concept model into corresponding Java representations. There are three types of concepts, each of which is treated in a different way:

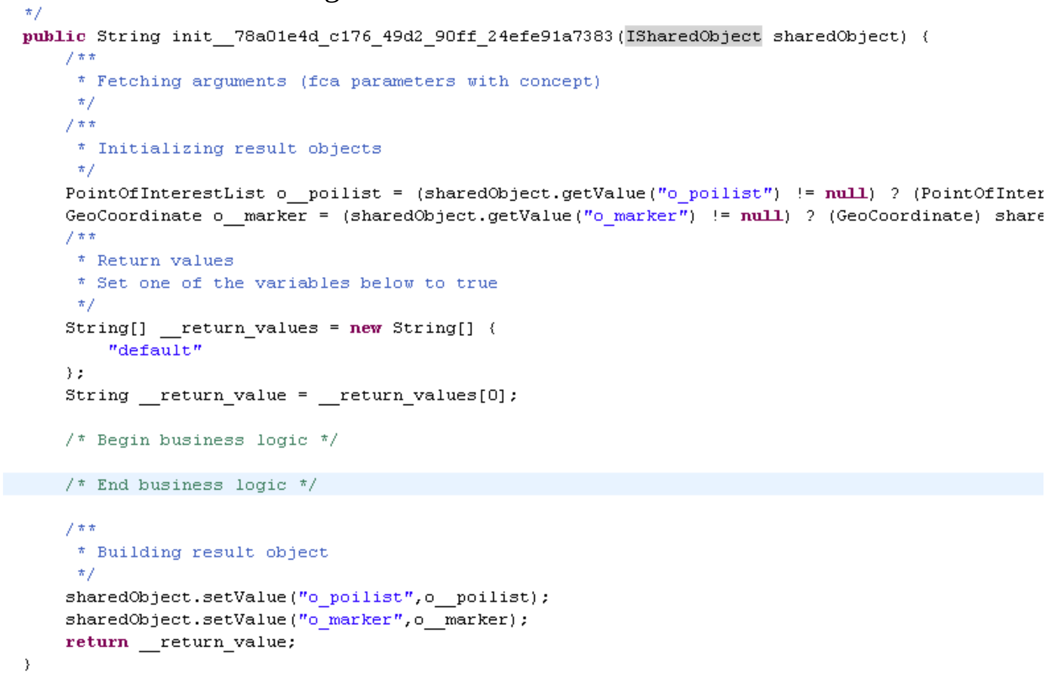
- *Complex concepts* are mapped to Java classes with member attributes for each of the concept's primitive or complex fields. These classes consist of an built-in reflection mechanism that allows direct manipulation of nested fields.
- *List concepts* are mapped to sub classes of *java.util.ArrayList*
- *Primitive concepts* are not actually transformed into code. Instead, existing Java primitive Types (e.g. Integer, Boolean) are used in place wherever primitive concepts are referenced from within other models

Generated artifacts:

- Each data type class is defined in a separate file under *de.emode.generated.datatypes*

3.2.4 FCA2Code

The FCA2Code transformation generates Java methods for all FCA methods. The code necessary for accessing the input and output parameters is already generated. The application developer is required to complement these methods with code that describes the actual business logic and initialize output parameters with the expected values. The Task Process Engine is responsible for calling the FCA methods, initializing the input parameter from the pin value of the associated system task and afterwards retrieving the result values from the method output parameters. All the information necessary to call the correct methods is already stored within the runtime task model and thus can be omitted during FCA2Code transformation. The developer is required to add her own code between the "begin business logic" and "end business logic" tags. Only then will it be preserved in case of repeated transformation. Figure 12 shows the runtime representation of an FCA method as it is created during transformation.



```
 */
public String init__78a01e4d_c176_49d2_90ff_24efe91a7383(ISharedObject sharedObject) {
    /**
     * Fetching arguments (fca parameters with concept)
     */
    /**
     * Initializing result objects
     */
    PointOfInterestList o__poilist = (sharedObject.getValue("o_poilist") != null) ? (PointOfInter
    GeoCoordinate o__marker = (sharedObject.getValue("o_marker") != null) ? (GeoCoordinate) share
    /**
     * Return values
     * Set one of the variables below to true
     */
    String[] __return_values = new String[] {
        "default"
    };
    String __return_value = __return_values[0];

    /* Begin business logic */

    /* End business logic */

    /**
     * Building result object
     */
    sharedObject.setValue("o_poilist",o__poilist);
    sharedObject.setValue("o_marker",o__marker);
    return __return_value;
}
```

Figure 12: Representation of FCA methods at runtime

Generated artifacts:

- *EmodeApplicationFca.java* – The class containing all the FCA methods

3.2.5 Context2Code

This transformation will generate skeleton classes for each context provider defined in the context model. All classes implement the context provider API as defined by the EMODE context service [Emode 2006c]. Similar to FCA methods, the developer is required to add the implementation of the context provider to the sections marked by the "begin business logic" and "end business logic" tags. This in particular involves the aggregation of raw context information that may come from any kind of sensor. Additionally, providers may also act as context consumers in order to provide derived (high-level) context. As with the FCA methods, these manual changes are preserved in case of repeated transformation. Aside from the context provider classes, a configuration file which defines the mapping between context source and context parameter is also generated during this transformation. The file is required by the EMODE context service in order to locate the appropriate context sources at runtime.

Generated artifacts:

- Each provider is specified in a separate file in *de.emode.generated.contextsources*
- *domainresources.xml* – the configuration file containing the context provider mappings

3.2.6 AUI2Code

Finally, the AUI2Code transformation generates a set of D3ML files that together make up the multimodal user interface description of the application. Usually one D3ML file represents one multimodal dialogue associated with an interaction task. In case of AUI refinement multiple version of the same dialogue may exist, resulting in multiple D3ML files. For this reason, also an index file is generated for each dialogue, containing information about which UI version is to be used in dependency of the currently active modalities. In addition to the files representing the user interface, several configuration files are also generated during transformation.

Generated artifacts:

- One index file for each multimodal dialogue (*.idx)
- One or more D3ML files for each multimodal dialogue (*.d3ml)
- *emode.css* – this file contains Cascading Style Sheet definitions and is by default included when rendering the visual part of the user interface
- *msc.properties* – this file contains configuration parameters for the MSC
- *modalities.xml* – this file contains information about the available modalities and allows the developer to manually enable or disable individual modalities

4 Appendix A: Bibliography

[Balme 2004] Balme, L.; Demeure, A.; Barralon, N.; Coutaz, J. & Calvary, G.:CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces., *in* 'EUSAI', pp. 291-302. **2004**.

[Czarnecki 2003] Czarnecki, K. & Helsen, S.: Classification of Model Transformation Approaches, *Proceedings of the 2nd OOPSLA workshop on Generative Techniques in the Context of Model-driven Architecture*. ACM Press, **2003**.

[Demeure 2005] Demeure, A.; Calvary, G.; Sottet, J. & Vanderdonkt, J.: A reference model for distributed user interfaces, *TAMODIA '05: Proceedings of the 4th international workshop on Task models and diagrams*, ACM Press, **2005**, 79-86.

[Emode 2006a] EMODE Consortium (Behring, A. Edtr.): German EMODE Meta Model, (TR-4), Technical report, Telecooperation Research Division, TU Darmstadt, Darmstadt, ISSN 1864-0516. **2006**.

[Emode 2006b] EMODE Consortium (Petter, A. Edtr.): Deliverable D2.3 – Modelltransformation, (TR-5), Technical report, Telecooperation Research Division, TU Darmstadt, Darmstadt, ISSN 1864-0516. **2006**.

[Emode 2006c] Hamann, T.; Hübsch, G. & Neumerkel, R.: D3.1, Part 1 – Spezifikation der EMODE Entwicklungsumgebung, (TR-6), Technical report, TU Dresden, Dresden, ISSN 1864-0516. **2006**.

[Florins 2006] Florins, M.: Graceful Degradation: a Method for Designing Multiplatform Graphical User Interfaces, PhD-Thesis, *Université catholique de Louvain*, **2006.**

[Limbourg 2004] Limbourg, Q.; Vanderdonckt, J.; Michotte, B.; Bouillon, L.; Florins, M. & Trevisan, D. Luyten, K.; Abrams, M.; Limbourg, Q. & Vanderdonckt, J. (ed.): UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces, *Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages"* (Gallipoli, May 25, 2004), **2004**, 55-62.

[Miller 2003] Miller, J. & Mukerji, J.: MDA Guide Version 1.0.1, OMG. **2003**.

[Mori 2005] Mori, G. & Paternó, F.: Automatic semantic platform-dependent redesign, *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence, ACM Press*, **2005**, 177-182.

[OMG 2005] OMG: MOF QVT Final Adopted Specification, **2005**

[OMG 2006] OMG: Meta Object Facility (MOF) Core Specification, OMG Available Specification, Version 2.0. **2006**

[OMG 2007] OMG: Unified Modeling Language: Infrastructure, OMG Available Specification formal/07-02-06, Version 2.1.1. **2007**.

[Paternò 1997] Paternò, F. und Mancini, C. und Meniconi, S., ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, *INTERACT `97: Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction,* 362-369, Chapman & Hall, Ltd., London **1997**.

[Paternò 2002] Paternò, F., Santoro, C., One Model, Many Interfaces, *Proceedings of CADUI 2002* Valenciennes, France, May **2002**.

[Schmid 2006] Schmid, K.; Kortuem, G. (ed.): The Self-Adaptation Problem in Software Specifications*, Workshop on Software Engineering Challenges for Ubiquitous Computing,* **2006**, 43-44.

[Sottet 2005] Sottet, J.; Calvary, G. & Favre, J.: Towards Model-driven Engineering of Plastic User Interfaces, *in* Andreas Pleuß; Jan Van den Bergh; Heinrich Hußmann & Stefan Sauer, ed.,'MDDAUI', CEUR-WS.org, **2005**.

[Sottet 2006] Sottet, J., Calvary, G. & Favre, J.: Models at Run-time for Sustaining User Interface Plasticity, *Proceedings of the Workshop Models at Run Time*, **2006.**

[Stanciulescu 2006] Stanciulescu, A.: A Transformational Approach for Developing Multimodal Web User Interfaces, PhD-Thesis, *Université Catolique de Louvin,* **2006.**

[UIMS-WS 1992] The UIMS Developers Workshop: A Metamodel for the Runtime Architecture of An Interactive System, *SIGCHI Bulletin* **24**(1). **1992**, 32-37.

5 Appendix B: Model-To-Model Transformations Documentation

The transformations presented are taken from EMODE on 31st of August and present the state of work at that date.

5.1.1 Transformation: Goal2Task

```
transformation GoalToTasks (goalmodel:EMODE, taskmodel:EMODE) {

top relation functionalGoalToInteractionTask {

            theName:String;

            enforce domain goalmodel g:EMODESpecific::Goals::FunctionalGoal {
                        name=determineValue(g.name,theName)
            };

            enforce domain taskmodel t:EMODESpecific::Task::TaskExecutionNode {
                        name=determineValue(t.name,theName),
                        isOfType=determineTypeValue(t.isOfType, EMODESpecific::Task::TaskNodeKind::TNT_Abstract),
                        theSupportingTaskRelationEnd=m:EMODESpecific::Task::TaskSupportsGoal {
                                    theSupportedGoalEnd=g
                        }
            };

            when{
      theName = if t.oclIsUndefined() then g.name
                        else if t.name.oclIsUndefined() then g.name
                            else t.name
                            endif
                        endif;
   }
}


top relation subGoalToTaskHierarchy {

            checkonly domain goalmodel supg:EMODESpecific::Goals::FunctionalGoal {
                        theSuperGoalEndRelation=sg:EMODESpecific::Goals::SubGoalOf {
                                    theSubGoalEndGoal=subg:EMODESpecific::Goals::FunctionalGoal {}
                        }
            };

            enforce domain taskmodel tg:EMODESpecific::Task::TaskDefinition {
                        theRealizationEnd=supt:EMODESpecific::Task::TaskExecutionNode {
                                    theRealizedElement=tg
                        },
                        isOfType=determineTypeValue(tg.isOfType, EMODESpecific::Task::TaskNodeKind::TNT_Abstract),
                        containedNode=subt:EMODESpecific::Task::TaskExecutionNode {}
            };

            when {
                        functionalGoalToInteractionTask(subg,subt);
                        functionalGoalToInteractionTask(supg,supt);
            }

}

/* Returns the defaultValue if modelElementValue is undefined, otherwise modelElementValue is returned.
   This query can be used to prevent overriding string values in a target model element
*/
query determineValue(modelElementValue : String, defaultValue : String) : String {
       if modelElementValue.oclIsUndefined()
          then defaultValue
          else modelElementValue
       endif
}
query determineTypeValue(modelElementValue : EMODESpecific::Task::TaskNodeKind, defaultValue :
EMODESpecific::Task::TaskNodeKind) : EMODESpecific::Task::TaskNodeKind {
      if modelElementValue.oclIsUndefined()
          then defaultValue
          else modelElementValue
      endif
}
}
```

5.1.2 Transformation: FCA2Task

```
transformation FCAToTask1 (fcamodel:EMODE, taskmodel:EMODE) {

key EMODESpecific::FunctionalCoreAdapter::FCACall {
            theCalledFCAMethod
};

key EMODESpecific::FunctionalCoreAdapter::FCACallResult {
            theParamAssoiationWithRealizedParam
};

key EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
            theParamAssoiationWithRealizedParam
};

key EMODESpecific::EMODECommons::ParameterAssociation {
            theRealizingParamWithParamAssoiation
};

top relation FCAMethodToFCACall {

            theName:String;

            checkonly domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCAMethod {
                        name = determineValue(f.name, theName)
            };

            enforce domain taskmodel t:EMODESpecific::FunctionalCoreAdapter::FCACall {
                        theCalledFCAMethod=f,
                        name = determineValue(t.name, theName)
            };

            when {
               theName= if f.oclIsUndefined() then f.name
                        else if t.name.oclIsUndefined() then f.name
                            else t.name
                            endif
                        endif;
            }

            where {
                        FCAMethodParametersToFCACallParameters(f,t);
                        FCAMethodResultsToFCACallResults(f,t);
            }
}

relation FCAMethodParametersToFCACallParameters {

            checkonly domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCAMethod {
                        theFCAMethodParameter=fpp:EMODESpecific::FunctionalCoreAdapter::FCAMethodParameter {}
            };

            enforce domain taskmodel t:EMODESpecific::FunctionalCoreAdapter::FCACall {
                        theFCAParameter = fp : EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
                                    theParamAssoiationWithRealizedParam=pas:EMODESpecific::EMODECommons::ParameterAssociation
{
                                                theRealizingParamWithParamAssoiation=fpp
                                    }
                        },
                        ParameterAssociations=pas
            };
```

```
		where {
			FCAMethodParam2FCACallParamAttribs(fpp, fp);
		}
}

relation FCAMethodResultsToFCACallResults {

		checkonly domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCAMethod {
			theFCAMethodResults=fpr:EMODESpecific::FunctionalCoreAdapter::FCAMethodResult {}
		};

		enforce domain taskmodel t:EMODESpecific::FunctionalCoreAdapter::FCACall {
			theFCAResult = fp : EMODESpecific::FunctionalCoreAdapter::FCACallResult {
				theParamAssoiationWithRealizedParam=pas:EMODESpecific::EMODECommons::ParameterAssociation
{
					theRealizingParamWithParamAssoiation=fpr
				}
			},
			ParameterAssociations=pas
		};

		where {
			FCAMethodResults2FCACallResultsAttribs(fpr, fp);
		}

}

relation FCAMethodParam2FCACallParamAttribs {

		theName:String;

		checkonly domain fcamodel fpp:EMODESpecific::FunctionalCoreAdapter::FCAMethodParameter {
			name = determineValue(fpp.name, theName)
		};

		enforce domain taskmodel fp:EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
			name = determineValue(fp.name, theName)
		};

		when {
		   theName= if fpp.oclIsUndefined() then fpp.name
			     else if fp.name.oclIsUndefined() then fpp.name
				 else fp.name
				 endif
			     endif;
		}

		where {
			FCAMethodParam2FCACallParamCon(fpp, fp);
		}
}

relation FCAMethodParam2FCACallParamCon {

		con:EMODESpecific::DomainConcept::Concept;

		checkonly domain fcamodel fpp:EMODESpecific::FunctionalCoreAdapter::FCAMethodParameter {
			theConceptedElementConceptEnd=con
		};

		enforce domain taskmodel fp:EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
			theConceptedElementConceptEnd=con
		};

}

relation FCAMethodResults2FCACallResultsAttribs {

		theName:String;

		checkonly domain fcamodel fpr:EMODESpecific::FunctionalCoreAdapter::FCAMethodResult {
			name = determineValue(fpr.name, theName)
```

```
            };

            enforce domain taskmodel fp:EMODESpecific::FunctionalCoreAdapter::FCACallResult {
                        name = determineValue(fp.name, theName)
            };

            when {
               theName= if fpr.oclIsUndefined() then fpr.name
                      else if fp.name.oclIsUndefined() then fpr.name
                          else fp.name
                          endif
                      endif;
            }

            where {
                        FCAMethodResults2FCACallResultsCon(fpr, fp);
            }
}

relation FCAMethodResults2FCACallResultsCon {

            con:EMODESpecific::DomainConcept::Concept;

            checkonly domain fcamodel fpr:EMODESpecific::FunctionalCoreAdapter::FCAMethodResult {
                        theConceptedElementConceptEnd=con
            };

            enforce domain taskmodel fp:EMODESpecific::FunctionalCoreAdapter::FCACallResult {
                        theConceptedElementConceptEnd=con
            };
}

query determineValue(modelElementValue : String, defaultValue : String) : String {
      if modelElementValue.oclIsUndefined()
         then defaultValue
         else modelElementValue
      endif
}


}
*****************************************************
transformation FCAToTask2 (fcamodel:EMODE, taskmodel:EMODE) {

key EMODESpecific::Task::TaskExecutionNode {
            theImplementingFCACall
};

key EMODESpecific::Task::EMODEOutputPin {
            theParamAssoiationWithRealizedParam
};

key EMODESpecific::Task::EMODEInputPin {
            theParamAssoiationWithRealizedParam
};

key EMODESpecific::EMODECommons::ParameterAssociation {
            theRealizingParamWithParamAssoiation
};

top relation FCAMethodToFCACall {

            theName:String;

            checkonly domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCACall {
                        name = determineValue(f.name, theName)
            };

            enforce domain taskmodel t:EMODESpecific::Task::TaskExecutionNode {
                        theImplementingFCACall=f,
                        isOfType=EMODESpecific::Task::TaskNodeKind::TNT_System,
                        name = determineValue(t.name, theName)
            };

            when {
               theName= if f.oclIsUndefined() then f.name
                      else if t.name.oclIsUndefined() then f.name
                          else t.name
                          endif
                      endif;
            }

            where {
                        FCACallParametersToInputPins(f,t);
                        FCACallResultsToOutputPins(f,t);
            }
}

relation FCACallParametersToInputPins {

            theName:String;

            checkonly domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCACall {
                        theFCAParameter=fpr:EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
                                    name = determineValue(fpr.name, theName)
                        }
            };

            enforce domain taskmodel t:EMODESpecific::Task::TaskExecutionNode {
                        theArgumentsOfTheAction = ta : EMODESpecific::Task::EMODEInputPin {
                                    theParamAssoiationWithRealizedParam=pas:EMODESpecific::EMODECommons::ParameterAssociation
{
                                                theRealizingParamWithParamAssoiation=fpr
                                    },
                                    name = determineValue(ta.name, theName)
                        },
                        ParameterAssociations=pas
            };

            when {
               theName= if fpr.oclIsUndefined() then fpr.name
                      else if ta.name.oclIsUndefined() then fpr.name
                          else ta.name
                          endif
                      endif;
            }

            where {
                        FCACallParametersToInputCon(fpr, ta);
            }
}

relation FCACallParametersToInputCon {

            con:EMODESpecific::DomainConcept::Concept;

            checkonly domain fcamodel fpr:EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
                        theConceptedElementConceptEnd=con
            };

            enforce domain taskmodel ta : EMODESpecific::Task::EMODEInputPin {
                        theConceptedElementConceptEnd=con
            };

}

relation FCACallResultsToOutputPins {

            theName:String;

            checkonly domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCACall {
                        theFCAResult=fpr:EMODESpecific::FunctionalCoreAdapter::FCACallResult {
                                    name = determineValue(fpr.name, theName)
                        }
            };

            enforce domain taskmodel t:EMODESpecific::Task::TaskExecutionNode {
                        theResultsOfTheAction = ta : EMODESpecific::Task::EMODEOutputPin {
                                    theParamAssoiationWithRealizedParam=pas:EMODESpecific::EMODECommons::ParameterAssociation
{
                                                theRealizingParamWithParamAssoiation=fpr
                                    },
                                    name = determineValue(ta.name, theName)
                        },
                        ParameterAssociations=pas
            };

            when {
               theName= if fpr.oclIsUndefined() then fpr.name
                        else if ta.name.oclIsUndefined() then fpr.name
                            else ta.name
                            endif
                        endif;
            }

            where {
                        FCACallResultsToOutputCons(fpr, ta);
            }
}

relation FCACallResultsToOutputCons {

            con:EMODESpecific::DomainConcept::Concept;

            checkonly domain fcamodel fpr:EMODESpecific::FunctionalCoreAdapter::FCACallResult {
                        theConceptedElementConceptEnd=con
            };

            enforce domain taskmodel ta : EMODESpecific::Task::EMODEOutputPin {
                        theConceptedElementConceptEnd=con
            };
}



/* Returns the defaultValue if modelElementValue is undefined, otherwise modelElementValue is returned.
   This query can be used to prevent overriding string values in a target model element
*/
query determineValue(modelElementValue : String, defaultValue : String) : String {
       if modelElementValue.oclIsUndefined()
          then defaultValue
          else modelElementValue
       endif
}
```

5.1.3 Transformation: Task2Dialoguespace

```
transformation GenerateDialogueSpaceModel (taskmodel:EMODE, dialoguespace:EMODE) {

key EMODESpecific::DialogueSpace::AUIBox {
            LongDescription
};

key EMODESpecific::Diagramming::AUIDiagram {
            notes
};

key EMODESpecific::DomainConcept::ConceptValueAccess {
            theAccessedValue
};

key EMODESpecific::DialogueSpace::AUIInteractor {
            theTaskNode2BeEnacted
};
```

```
/* ---------------------------------------------------------------------- */
/* Top Level AUIBox erzeugen, wird immer erzeugt, aber nicht immer benötigt */
top relation generateToplevelAUI {

        checkonly domain taskmodel f:EMODESpecific::Diagramming::TaskDiagram {};

        enforce domain dialoguespace b:EMODESpecific::DialogueSpace::AUIBox {
                name=determineValue(b.name, 'RootAUIBox'),
                theDefiningDiagramOfTheElement=w:EMODESpecific::Diagramming::AUIDiagram {
                        name=determineValue(w.name, 'RootAUIBox'),
                        notes='RootAUIBox'
                },
                Description='trafo',
                LongDescription='RootAUIBox'
        };

}

/* ---------------------------------------------------------------------- */
/* Aufregende Defintion der Transformation von taks in AUIInteraktoren */
top relation taskdefinitionToAUIInteractor {

        theName:String;
        q:EMODESpecific::DomainConcept::RDF::RDFBase::UniformResourceIdentifier;

        enforce domain taskmodel f:EMODESpecific::Diagramming::TaskDiagram {
                theElementWithDefiningDiagram=t:EMODESpecific::Task::TaskDefinition {
                        name=determineValue(t.name,theName)
                }
        };

        enforce domain dialoguespace d:EMODESpecific::DialogueSpace::AUIInteractor {
                theNamespaceOfTheElements=b:EMODESpecific::DialogueSpace::AUIInteractor {},
                theOwningInstance=b:EMODESpecific::DialogueSpace::AUIInteractor {},
                owner=b:EMODESpecific::DialogueSpace::AUIInteractor {},
                name=determineValue(d.name,theName),
                referencedURI=q:EMODESpecific::DomainConcept::RDF::RDFBase::UniformResourceIdentifier {
                        name=determineValue(q.name, 'http://www.emode-projekt.de/d3mlLib/0.1/container')
                },
                theDefinitionUsingTheAUI=t
        };

        when {
                q=searchDefinitionUri(t);
                generateToplevelAUI(f, b);
                b =                     if hasCorrectAUIBox4Definition(t) then searchCorrectAUIBox4Definition(t)
                                                else b
                                                endif;
          theName =             if d.oclIsUndefined() then t.name
                else
                        if d.name.oclIsUndefined() then t.name
                        else d.name
                        endif
                endif;
        }

        where {
                d=searchAUIInteractor4Definition(t);
                seturiinfo(d,q);
        }

}

/* ---------------------------------------------------------------------- */
/* Top Level AUIBox erzeugen, wird immer erzeugt, aber nicht immer benötigt */
top relation executableTaskNodeToAUIInteractor {

        theName:String;
        q:EMODESpecific::DomainConcept::RDF::RDFBase::UniformResourceIdentifier;

        enforce domain taskmodel t:EMODESpecific::Task::TaskExecutionNode {
                isOfType=EMODESpecific::Task::TaskNodeKind::TNT_Interaction,
                name=determineValue(t.name,theName) ,
                theVisualizationOfTheElement=e:EMODESpecific::Diagramming::ElementRepresentation {
                        theCanvasWithElements=f:EMODESpecific::Diagramming::TaskDiagram {}
                }
        };

        enforce domain dialoguespace d:EMODESpecific::DialogueSpace::AUIInteractor {
                theNamespaceOfTheElements=b:EMODESpecific::DialogueSpace::AUIInteractor{},
                theOwningInstance=b:EMODESpecific::DialogueSpace::AUIInteractor{},
                owner=b:EMODESpecific::DialogueSpace::AUIInteractor{},
                name=determineValue(d.name,theName),
                referencedURI=q:EMODESpecific::DomainConcept::RDF::RDFBase::UniformResourceIdentifier {
                        name=determineValue(q.name, 'http://www.emode-projekt.de/d3mlLib/0.1/container')
                },
                theTaskNode2BeEnacted=t
        };

        when {
                q=searchUri(t);
                taskdefinitionToAUIInteractor(f, b) or generateToplevelAUI(f, b);
                b =                     if hasCorrectAUIBox(t) then searchCorrectAUIBox(t)
                                                else b
                                                endif;
          theName =             if d.oclIsUndefined() then t.name
                else
                        if d.name.oclIsUndefined() then t.name
                        else d.name
                        endif
                endif;
        }

        where {
                seturiinfo(d,q);
                inputPinsToSubInteractors(t, d);
                outputPinsToSubInteractors(t, d);
                /* generateTopAUIInteractorForForks();
                generateAUISpaces(); */
        }
}

/* ---------------------------------------------------------------- */
/* Korrekte URi setzen           */
relation seturiinfo {

 enforce domain taskmodel d:EMODESpecific::DialogueSpace::AUIInteractor {};

 enforce domain dialoguespace q:EMODESpecific::DomainConcept::RDF::RDFBase::UniformResourceIdentifier {
        name=determineURIValue(q.name,'http://www.emode-projekt.de/d3mlLib/0.1/container')
 };
}

/* ---------------------------------------------------------------- */
/* AUI Interaktoren aus den entsprechenden Pins erzeugen in/output */
relation inputPinsToSubInteractors {

        theName:String;
        da:EMODESpecific::DialogueSpace::AUIInteractor;

        checkonly domain taskmodel t:EMODESpecific::Task::TaskExecutionNode {
                theArgumentsOfTheAction = pi:EMODESpecific::Task::EMODEInputPin {
                        name=determineValue(pi.name,theName)
                }
        };


        enforce domain dialoguespace d:EMODESpecific::DialogueSpace::AUIInteractor {
                theInstanceOwnedInstances = da:EMODESpecific::DialogueSpace::AUIInteractor {
                        name=determineValue(da.name,theName),
                        theElementsConceptValueAccess=c:EMODESpecific::DomainConcept::ConceptValueAccess {
                                theAccessedValue=pi
                        }
                        },
                        ownedElements = da,
                        theElementsOfTheNamespace = da
            };

            when {
                        da=searchSubAUIInteractorPin(pi);
               theName = if pi.oclIsUndefined() and not da.oclIsUndefined() then da.name
                        else if pi.name.oclIsUndefined() and not da.name.oclIsUndefined() then da.name
                            else pi.name
                            endif
                        endif;
            }
}

relation outputPinsToSubInteractors {

            theName:String;
            da:EMODESpecific::DialogueSpace::AUIInteractor;

            checkonly domain taskmodel t:EMODESpecific::Task::TaskExecutionNode {
                        theResultsOfTheAction = pi:EMODESpecific::Task::EMODEOutputPin {
                                    name=determineValue(pi.name,theName)
                        }
            };


            enforce domain dialoguespace d:EMODESpecific::DialogueSpace::AUIInteractor {
                        theInstanceOwnedInstances = da:EMODESpecific::DialogueSpace::AUIInteractor {
                                    name=determineValue(da.name,theName),
                                    theElementsConceptValueAccess=c:EMODESpecific::DomainConcept::ConceptValueAccess {
                                                theAccessedValue=pi
                                    }
                        },
                        ownedElements = da,
                        theElementsOfTheNamespace = da
            };

            when {
                        da=searchSubAUIInteractorPin(pi);
               theName = if pi.oclIsUndefined() and not da.oclIsUndefined() then da.name
                        else if pi.name.oclIsUndefined() and not da.name.oclIsUndefined() then da.name
                            else pi.name
                            endif
                        endif;
            }
}

query determineValue(modelElementValue : String, defaultValue : String) : String {
       if modelElementValue.oclIsUndefined()
          then defaultValue
          else modelElementValue
       endif
}

query determineURIValue(modelElementValue : String, defaultValue : String) : String {
       if modelElementValue.oclIsUndefined()
          then defaultValue
          else if modelElementValue='' then
            defaultValue
            else modelElementValue
            endif
       endif
}

query
searchSubAUIInteractorPin(t:EMODESpecific::EMODECommons::EMODEConceptedElement):EMODESpecific::DialogueSpace::AUIInter
actor {
         EMODESpecific::DialogueSpace::AUIInteractor.allInstances()->select(x|x.theElementsConceptValueAccess-
>exists(c|c=((EMODESpecific::DomainConcept::ConceptValueAccess.allInstances()->select(y|y.theAccessedValue=t))-
>asSequence()->first())))->asSequence()->first()
}
```

```
query
searchUri(t:EMODESpecific::Task::TaskExecutionNode):EMODESpecific::DomainConcept::RDF::RDFBase::UniformResourceIdentifier {
	EMODESpecific::DialogueSpace::AUIInteractor.allInstances()->select(x|x.theTaskNode2BeEnacted=t)->asSequence()->first().referencedURI
}

query
searchDefinitionUri(t:EMODESpecific::Task::TaskDefinition):EMODESpecific::DomainConcept::RDF::RDFBase::UniformResourceIdentifier {
	EMODESpecific::DialogueSpace::AUIInteractor.allInstances()->select(x|x.theDefinitionUsingTheAUI=t)->asSequence()->first().referencedURI
}

query
searchCorrectAUIBox(t:EMODESpecific::Task::TaskExecutionNode):EMODESpecific::EMODECommons::Classes::EMODEInstanceSpecification {
	EMODESpecific::DialogueSpace::AUIInteractor.allInstances()->select(x|x.theTaskNode2BeEnacted=t)->asSequence()->first().theOwningInstance
}

query hasCorrectAUIBox(t:EMODESpecific::Task::TaskExecutionNode):Boolean {
	not(searchCorrectAUIBox(t).oclIsUndefined())
}

query
searchCorrectAUIBox4Definition(t:EMODESpecific::Task::TaskDefinition):EMODESpecific::EMODECommons::Classes::EMODEInstanceSpecification {
	EMODESpecific::DialogueSpace::AUIInteractor.allInstances()->select(x|x.theDefinitionUsingTheAUI=t)->asSequence()->first().theOwningInstance
}

query hasCorrectAUIBox4Definition(t:EMODESpecific::Task::TaskDefinition):Boolean {
	not(searchCorrectAUIBox4Definition(t).oclIsUndefined())
}

query searchAUIInteractor4Definition(t:EMODESpecific::Task::TaskDefinition):EMODESpecific::DialogueSpace::AUIInteractor {
	EMODESpecific::DialogueSpace::AUIInteractor.allInstances()->select(x|x.theDefinitionUsingTheAUI=t)->asSequence()->first()
}

query searchAUIInteractorForInteractor(t:EMODESpecific::Task::TaskExecutionNode):EMODESpecific::DialogueSpace::AUIInteractor
{
	searchAUIInteractor4Definition(EMODESpecific::Task::TaskDefinition.allInstances()->select(x|x.containedNode->exists(y|y=t))->asSequence()->first())
}

query hasAUIInteractorForInteractor(t:EMODESpecific::Task::TaskExecutionNode):Boolean {
	not(searchAUIInteractorForInteractor(t).oclIsUndefined())
}

query _debug(t:EMODESpecific::DialogueSpace::AUIInteractor):EMODESpecific::DialogueSpace::AUIInteractor {
	t
}


}
```

5.1.4 Transformation: Task2FCA

```
transformation TaskToFCA1 (taskmodel:EMODE, fcamodel:EMODE) {

key EMODESpecific::FunctionalCoreAdapter::FCACall {
	theImplementedTaskEnd
};

key EMODESpecific::FunctionalCoreAdapter::FCACallResult {
	theParamAssoiationWithRealizingParam
};

key EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
	theParamAssoiationWithRealizingParam
```

```
};

key EMODESpecific::EMODECommons::ParameterAssociation {
        theRealizedParamOfTheParamAssociation
};

top relation taskExecutionNodeToFCACall {

        theName:String;

        checkonly domain taskmodel t:EMODESpecific::Task::TaskExecutionNode {
                name = determineValue(t.name, theName),
                isOfType=EMODESpecific::Task::TaskNodeKind::TNT_System
        };

        enforce domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCACall {
                name = determineValue(f.name, theName),
                theImplementedTaskEnd=t
        };

        when {
                /* t.theImplementingFCACall.oclIsUndefined(); */
                /* f=searchFCACall(t); */
          theName= if t.oclIsUndefined() then t.name
               else if f.name.oclIsUndefined() then t.name
                  else f.name
                  endif
               endif;
                t.theRealizedTaskEnd->isEmpty();
        }

        where {
                taskExecutionNodeToFCACallInputParameters(t, f);
                taskExecutionNodeToFCACallOutputParameters(t, f);
        }
}

relation taskExecutionNodeToFCACallInputParameters {

        checkonly domain taskmodel t:EMODESpecific::Task::TaskExecutionNode {
                theArgumentsOfTheAction = ta : EMODESpecific::Task::EMODEInputPin {}
        };

        enforce domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCACall {
                theFCAParameter = fp : EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
                        theParamAssoiationWithRealizingParam=pa:EMODESpecific::EMODECommons::ParameterAssociation {
                                theRealizedParamOfTheParamAssociation=ta:EMODESpecific::Task::EMODEInputPin {
                                        theActionWithArguments=t:EMODESpecific::Task::TaskExecutionNode {
                                                ParameterAssociations=pa
                                        }
                                }
                        }
                }
        };

        where {
                emodeInputPinToFcacallParameter(ta, fp);
        }

}

relation taskExecutionNodeToFCACallOutputParameters {

        checkonly domain taskmodel t:EMODESpecific::Task::TaskExecutionNode {
                theResultsOfTheAction = ta : EMODESpecific::Task::EMODEOutputPin {}
        };

        enforce domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCACall {
                theFCAResult = fp : EMODESpecific::FunctionalCoreAdapter::FCACallResult {
                        theParamAssoiationWithRealizingParam=pa:EMODESpecific::EMODECommons::ParameterAssociation {
                                theRealizedParamOfTheParamAssociation=ta:EMODESpecific::Task::EMODEOutputPin {
                                        theActionWithResults=t:EMODESpecific::Task::TaskExecutionNode {
                                                ParameterAssociations=pa
                                          }
                                    }
                              }
                        }
                  };

                  where {
                        emodeOutputPinToFcacallResult(ta, fp);
                  }

}

relation emodeInputPinToFcacallParameter {

                  theName:String;

                  checkonly domain taskmodel ta:EMODESpecific::Task::EMODEInputPin {
                        name = determineValue(ta.name, theName),
                        theActionWithArguments=tsk:EMODESpecific::Task::TaskExecutionNode {}
                  };

                  enforce domain fcamodel fp:EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
                        name = determineValue(fp.name, theName)
                  };

    when{
       theName = if fp.oclIsUndefined() then ta.name
                        else if fp.name.oclIsUndefined() then ta.name
                            else fp.name
                            endif
                        endif;
    }

    where {
                  emodeInputPinToFcacallParCon(ta, fp);
    }
}

relation emodeInputPinToFcacallParCon {

                  con:EMODESpecific::DomainConcept::Concept;

                  checkonly domain taskmodel ta:EMODESpecific::Task::EMODEInputPin {
                        theConceptedElementConceptEnd=con
                  };

                  enforce domain fcamodel fp:EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
                        theConceptedElementConceptEnd=con
                  };
}


relation emodeOutputPinToFcacallResult {

                  theName:String;

                  checkonly domain taskmodel tr:EMODESpecific::Task::EMODEOutputPin {
                        name = determineValue(tr.name, theName),
                        theActionWithResults=tsk:EMODESpecific::Task::TaskExecutionNode {}
                  };

                  enforce domain fcamodel fp:EMODESpecific::FunctionalCoreAdapter::FCACallResult {
                        name = determineValue(fp.name, theName)
                  };

                  when{
       theName = if fp.oclIsUndefined() then tr.name
                        else if fp.name.oclIsUndefined() then tr.name
                            else fp.name
                            endif
                        endif;
    }

    where {
            emodeOutputPinToFcacallResConcept(tr, fp);
    }
}

relation emodeOutputPinToFcacallResConcept {

            con:EMODESpecific::DomainConcept::Concept;

            checkonly domain taskmodel tr:EMODESpecific::Task::EMODEOutputPin {
                        theConceptedElementConceptEnd=con
            };

            enforce domain fcamodel fp:EMODESpecific::FunctionalCoreAdapter::FCACallResult {
                        theConceptedElementConceptEnd=con

            };

}


query determineValue(modelElementValue : String, defaultValue : String) : String {
        if modelElementValue.oclIsUndefined()
           then defaultValue
           else modelElementValue
        endif
}

}
*****************************************************
transformation TaskToFCA2 (taskmodel:EMODE, fcamodel:EMODE) {

key EMODESpecific::FunctionalCoreAdapter::FCAMethod {
            theCallingFCACall
};

key EMODESpecific::FunctionalCoreAdapter::FCAMethodResult {
            theParamAssoiationWithRealizingParam
};

key EMODESpecific::FunctionalCoreAdapter::FCAMethodParameter {
            theParamAssoiationWithRealizingParam
};

key EMODESpecific::EMODECommons::ParameterAssociation {
            theRealizedParamOfTheParamAssociation
};

top relation FCACallToFCAMethod {

            theName:String;

            checkonly domain taskmodel t:EMODESpecific::FunctionalCoreAdapter::FCACall {
                        name = determineValue(t.name, theName)
            };

            enforce domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCAMethod {
                        name = determineValue(f.name, theName),
                        theCallingFCACall=t
            };

            when {
               theName= if t.oclIsUndefined() then t.name
                        else if f.name.oclIsUndefined() then t.name
                            else f.name
                            endif
                        endif;
            }

            where {
                    FCACallParameterToFCAMethodParameters(t, f);
                    FCACallResultToFCAMethodResults(t, f);
          }
}

relation FCACallParameterToFCAMethodParameters {

          checkonly domain taskmodel t:EMODESpecific::FunctionalCoreAdapter::FCACall {
                    theFCAParameter = ta : EMODESpecific::FunctionalCoreAdapter::FCACallParameter {}
          };

          enforce domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCAMethod {
                    theFCAMethodParameter = fp : EMODESpecific::FunctionalCoreAdapter::FCAMethodParameter {
                              theParamAssoiationWithRealizingParam=pa:EMODESpecific::EMODECommons::ParameterAssociation {

          theRealizedParamOfTheParamAssociation=ta:EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
                                                  theFCAWithParameters=t:EMODESpecific::FunctionalCoreAdapter::FCACall {
                                                            ParameterAssociations=pa
                                                  }
                                        }
                              }
                    }
          };

          where {
                    FCACallParameterToFCAMethodParameterProperties(ta, fp);
          }

}

relation FCACallResultToFCAMethodResults {

          checkonly domain taskmodel t:EMODESpecific::FunctionalCoreAdapter::FCACall {
                    theFCAResult = ta : EMODESpecific::FunctionalCoreAdapter::FCACallResult {}
          };

          enforce domain fcamodel f:EMODESpecific::FunctionalCoreAdapter::FCAMethod {
                    theFCAMethodResults = fp : EMODESpecific::FunctionalCoreAdapter::FCAMethodResult {
                              theParamAssoiationWithRealizingParam=pa:EMODESpecific::EMODECommons::ParameterAssociation {

          theRealizedParamOfTheParamAssociation=ta:EMODESpecific::FunctionalCoreAdapter::FCACallResult {
                                                  theFCAWithResults=t:EMODESpecific::FunctionalCoreAdapter::FCACall {
                                                            ParameterAssociations=pa
                                                  }
                                        }
                              }
                    }
          };

          where {
                    FCACallResultToFCAMethodResultProperties(ta, fp);
          }

}

relation FCACallParameterToFCAMethodParameterProperties {

          theName:String;


          checkonly domain taskmodel ta:EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
                    name = determineValue(ta.name, theName)
          };

          enforce domain fcamodel fp:EMODESpecific::FunctionalCoreAdapter::FCAMethodParameter {
                    name = determineValue(fp.name, theName)
          };

   when{
      theName = if fp.oclIsUndefined() then ta.name
                    else if fp.name.oclIsUndefined() then ta.name
                         else fp.name
```

```
                          endif
                      endif;
    }

    where {
            FCACallParameterToFCAMethodParameterPropCon(ta, fp);
    }
}

relation FCACallParameterToFCAMethodParameterPropCon {

            con:EMODESpecific::DomainConcept::Concept;

            checkonly domain taskmodel ta:EMODESpecific::FunctionalCoreAdapter::FCACallParameter {
                        theConceptedElementConceptEnd=con
            };

            enforce domain fcamodel fp:EMODESpecific::FunctionalCoreAdapter::FCAMethodParameter {
                        theConceptedElementConceptEnd=con
            };
}


relation FCACallResultToFCAMethodResultProperties {

            theName:String;

            checkonly domain taskmodel tr:EMODESpecific::FunctionalCoreAdapter::FCACallResult {
                        name = determineValue(tr.name, theName)
            };

            enforce domain fcamodel fp:EMODESpecific::FunctionalCoreAdapter::FCAMethodResult {
                        name = determineValue(fp.name, theName)
            };

            when{
        theName = if fp.oclIsUndefined() then tr.name
                      else if fp.name.oclIsUndefined() then tr.name
                          else fp.name
                          endif
                      endif;
    }

    where {
            FCACallResultToFCAMethodResultPropCon(tr, fp);
    }
}

relation FCACallResultToFCAMethodResultPropCon {

            con:EMODESpecific::DomainConcept::Concept;

            checkonly domain taskmodel tr:EMODESpecific::FunctionalCoreAdapter::FCACallResult {
                        theConceptedElementConceptEnd=con
            };

            enforce domain fcamodel fp:EMODESpecific::FunctionalCoreAdapter::FCAMethodResult {
                        theConceptedElementConceptEnd=con
            };

}



query determineValue(modelElementValue : String, defaultValue : String) : String {
       if modelElementValue.oclIsUndefined()
          then defaultValue
          else modelElementValue
       endif
}
```

```
}
```

6 Appendix C: Metamodel Changelog

This appendix includes a summary of the changes that have been made since deliverable D2.2 (version 1,0) [Emode 2006a]. It is a copy of the record that was constantly updated while modifying the metamodel. On top, the latest changes are documents (1.18 was released on 6th of July 2007).

6.1 Changes to Version 1.18

- Bayes ContextProvider support
- Ruler-Guides Revision

6.2 Changes to version 1.17

- *Support for ManipulationConcepts*:
  - new element ManipulatingElement derived from OWLClass
  - ManipulationConcept can be connected to a ManipulationConcept (ManipulationConcepts can be attached via uriRefs or direct)
  - TaskExecutionNode and AUIComponent are derived from ManipulatingElement
- TaskRelation removed
- Properties "RealizingParameter" und "RealizedParameter" removed in favor of associations (between ParameterAssociation and ParamTypeElement) of the same name
- Set ParamOverloadableElement.inputParameter and .outputParameter multiplicity to 0..n
- *RulerGuide Support*:
  - new element RulerGuide with associations HorizontalRulerGuide and VerticalRulerGuide
  - new constraint RulerGuide_1
- *Situations available for context-integration*:
  - NeeedModalityRequirements now points to a Situation
  - Situation derived from OWLClass
  - New element ContextQueryingElement capsules the query to the context service
  - New situation "Queried Situation", dervied from Situation and ContextQueryingElement
  - Elements TaskExecutionNode, ModalityRequirementProfile, AUIComponent are derived from Situation
- Associations theSupportingTask and theSupportedGoal set bidirectionally navigable
- *AUIHierarchy now covered by AUI Model - not Task Model*:
  - New association InteractorsInteractionSpace between AUISpace and AUIInteractor
- *URIInstanciation now via uniformResourceIdentifier*:
  - URIInstanciation is not associated to an URIReference anymore but has a property named "referencedURI" of type "UniformResourceIdentifier", multiplicity 0..1
  - Updated constraints on diagram URIUsage: EMODEInstanceSpecification*1, EMODEInstanceSpecification*2, EMODESlot_1
- Constraint "TaskDefinition_3" in diagram "TaskDefinitions" updated to reflect current semantic of abstract node type
- *Support for Runtime Voice (Cursor-Steuerung)*:
  - new association "auiElementOrdering"
- *Support for AUI Refinement*:
  - Rules can be formulated using the "RuleStatement" element that can be associated to named elements (being the parameters of the rule)

- EMODEOwnership association composition moved to "owner"-end
- EMODEOwnership association set bidirectionally navigable
- *Support for events*:
  - new Elements EventProvider, EventConsumer (abstract)
  - new Enumeration EventProviderStateChange
  - Removed ContextPush, ContextDataUse, ContextOutputQueryPin
  - Renamed ContextQuery to ContextEventProvider and derived it from EventProvider and ContextQueryingElement
  - New Elements MessageEndConnector and MessageEndDefinition
  - New elements ConceptObserverEventProvider and ConceptDelegationEventConsumer
  - New elements EventConsumerInitialNode and EventProviderFinalNode with enum EPFinalNodeTriggers
  - Attribute EMODEPin.MessagingPin renamed to isOfMessagingType and moved to ParamTypeElement, default set to "false"
  - Constraint ParamOverloadableElement_5 on diagram Parameterization updated
- Associations DomainForProperty and RangeForProperty set to bidirectionally navigable
- *Support for Pin semantics*:
  - new element "ParamTypeSemantic" defining semantics for parameters
  - new association (and URIReference attribute) "ParamSemanticsType" to reference a semantic
- Constraint EMODESlot*2 on diagram URIUsage removed and EMODESlot*1 modified
- Constraints in diagram Parameteriyation received names: ParamOverloadableElement*6 to ParamOverloadableElement*8
- Constraint ConceptNode_1 on diagram TasKNodes updated
- EMODEProperty.isReadOnly initial value set to "false"
- EMODEProperty Generalization Association ends renamed (spelling error)
- new element URIReferenceAlternative with constraint URIReferenceAlternative_1 on diagram URIReferencing

6.3 Changes to version 1.164

- *TaskNodeType "abstract" added*:
  - added TNT_Abstract to enumTaskNodeTypes
  - modified constraints TaskDefinition*3, TaskExecutionNode*8
- New operation "delete" for EMODEElement
- Multiplicity of AUIComponentRefined association ends set to 0..n, ordered
- New association AUIRelationRefined that associates an AUIComponentRelation with itself
- New element "Situation" with probability of occurance
- New element SituationImplication that can imply enumSituationImplication on an element
- New attributes "ProbabilityForEnteringInformation" and "ProbabilityForInformationPerception" for the AUIInteractor
- New attribute "MessagingPin" for EMODEPin

6.4 Changes to version 1.163

- *Elements had a derivation from EMODEElement added*:
  - ConceptValueAccess
  - Coordinate

    - ParameterAssociation
    - ParamOverloadableElement
    - LibSpecialToolAttributes
    - PatternParticipation
    - Binding
- *Removed*:
    - EMODEParameter
    - GroupableTaskNode
- PatternParticipation element now contains PatternParticipations
- *New classes derived from EMODEAssociation*:
    - PropertyGenerlization to generalize an EMODEProperty
    - PropertyEquivalence to depict equivalence of properties
    - ClassifierEquivalence to depict equivalence of classifiers
    - Rename EMODEClassifier's General association to ClassifierGeneral
- GeneralizationGeneral and GeneralizationSpecial for EMODEClassifiers set to binavigabel
- (Classifier)General for EMODEClassifiers set to binavigabel

6.5 Changes to version 1.162

- In Trafo-Package set association ends to ordered:
    - ClassifierValue.theValueReference
    - Bindings.binding
    - BoundVariableValue: variableValue instead of binding end
    - TraceKey.values

6.6 Changes to version 1.161

- TaskExecutionNode.input and .output multiplicities changed to 0..*
- Association OwnedInstances between EMODEInstanceSpecification adn EMODEClassifier is renamed to ClassOwnedInstances (as well as ist ends)
- A new association InstanceOwnedInstances is created circular between EMODEInstanceSpecification
- EMODEInstanceSpecification is derived from EMODENameSpace
- constraints EMODEInstanceSpecification_3, _4 and _5 added to the diagram "Composition"
- diagram TaskInheritancesOther is renamed to TaskDefinitions
- TaskExecutionNode set to non-abstract
- Constraint TaskExecutionNode_3 (diagram TaskToDialogueSpace) reformulated
- Constraints EMODEOutputPin*1 and EMODEInputPin*1 (diagram TaskPins) reformulated
- Association TaskRealization is reconnected from TaskBehaviorExecution to TaskExecutionNode
- Constraint StructuredTaskExecution_2 removed
- Constraint EndlessTask_2 (diagram TaskNodeTypes) reformulated
- TaskEdge.NecessaryEdge is removed
- Constraints InitialNode*2 reformulated and new constraint FinalNode*2 (diagram TaskControl)
- *diagram BasicTaskNodes:*
    - Constraint TaskDefinition*1 renamed to TaskDefintion*5 and moved to diagram TaskDefinitions

  - Constraints TaskExecutionNode*1 and StructuredTaskExecution*1 removed
- *diagram TaskEdges:*
  - Constraint TaskEdge*2 is reformulated*
  - *Constraints TaskObjectEdge*3 reformulated
- *diagram TaskDefinitions:*
  - Constraint StructuredTaskExecution_2 is removed
  - Constraint TaskDefinition*2 is renamed to TaskDefinition*3
- *diagram TaskExecutionOfBehavior:*
  - Association TaskImplementation is rerouted from TaskBehaviorExecution to TaskExecutionNode
  - Constraint TaskBehaviorExecution*1 and 2 are renamed to TaskExecutionNode7 and _8 and reformulated*
  - *Constraint TaskBehaviorExecution*3 is removed
  - Derivation TaskBehaviorExecution*4 is renamed to TaskExecutionNode*9 and (wording cleared up only) reformulated
  - Derivation TaskDefinition*1 is renamed to TaskDefinition*4 and moved to diagram TaskDefinitions
- *diagram TaskPinUsage:*
  - Constraint UserTask*1 renamed to TaskExecutionNode*4 and reformulated
  - Derivation TaskBehaviorExecution*1 and 2 reformulted to TaskExecutionNode5 and _6 respectively*
  - *Derivation TaskDefinition*1 and _2 reformulated and moved to diagram TaskDefinitions
- New diagram ConceptAccess with element ConceptValueAccess and ist associations to EMODEConceptedElement and EMODEElement
- Association AssociatedSubDiagram renamed to Defining Diagram and rerouted from DiagramElement to EMODElement

6.7 Changes to version 1.16

- ModalityRestrictions final refactoring to ModalityRequirements
  - cf. Title "AUI" - name fixed
  - TaskEdge note was changed
  - ModalityRestrictionProfile to ModalityRequirementsProfile
  - ModalityRestrictionProperty to ModalityRequirementsProperty
  - ModalityRestrictionProperty.Restriction to …Requirement
  - Associations ModalityRestrictionGeneralization & RestrictionDefinitions with their ends
- Lot of elements have comment & description attributes removed and inheritance to DescribedElement implemened instead. Among these:
  - Goal
  - TaskExecutionNode
  - NamedElement
  - ModelDescription
- Diagram object.isLocked, TaskNodeGroup.FastExit, TaskDefinition.FastExit, TaskEdge.NecessaryEdge defaults are lower-cased
- Assoziation URIRefForResource (RDFSResource - URIReference) set bi-directional
- Classification of AUIComponentRelations
  - New Diagram AUIComponentRelationTypes
  - AUIComponentRelation now is an EMODEInstanceSpecification
  - New element AUIComponentRelationClassifier

- AUIComponentRelation now is not abstract anymore
- AUIComponentRelationClassifier is now supertype of Binary, Unary and NAry AUIComp.Relations

- Changes RelatedAUIComponents.theComponents to [1..*]
- new attribute "ProviderConfiguration" in ContextProvider Element
- Change Task-Typing to enumeration rather than subclassing (*Bugzilla Bug 3*)
    - Deleted Endless Task and its subclasses
    - Set isOfType in TaskExecutionNode & TaskDefinition to "not derived"
    - is Endless is removed from TaskDefinition
    - Rearranged Diagrams BasicTaskNodes & TaskNodeTypes
    - Restated association "TaskImplementation", which previously connected SystemTask & FCACall. Now it connects TaskBehaviorExecution and FCACall; changed assocEnd name "theTaskImplementationEnd" to "theImplementingFCACall"
    - Constraint "TaskNodeGroup*1" in diagram TaskInheritancesOther is reformulated to "TaskDefinition*2" and "StructuredTaskExecution_2"
- Association "argument" between TaskExecutionNode and EMODEInputPin is now bidirectional (*Bugzilla Bug 2*)
- Link between Developer Association & EMODE Association restated
- Set to abstract: EMODEConceptedElement, EMODEClassifier, EMODEInstanceSpecification, EMODEProperty, EMODEDescribedElement, EMODENameSpace

6.8 Changes to version 1.153

- Created AUIInteractorClassifier, which specialize EMODEClassifier
- Changes superclass of AUIInteractor to EMODEInstanceSpecification
- Created new constraint "AUIInteractorClassifier_1"
- Supertype of EMODEModel set to Package (was Namespace)
- AUIInteractor.NeededModalityRequirements & InternalComponentsRelation converted to associations
- Changed EMODEProperty.defaultValue to EMODEValueSpecification Type

6.9 Changes to version 1.152

- Constraints SystemTask*1 and UserTask*1 are removed and instead, derivation ExecutableTaskNodeGroup_1 introduced
- Constraints moved between diagrams
- loaded Thomas' Domain Concept RC4 package
- RDFSResource was generalized to Concept
- Derived URIReference, NamespaceDefinition, LocalName and RDFNamespace from EMODEElement
- rename Value to M3Value, ClassifierValue to M3ComplexValue and PrimitiveValue to M3PrimitiveValue
- M3PrimitiveValue set as abstract
- ClassifierValue.classifier set to non-navigable
- EMODEPrimitives set as generalizations of the Trafo-Primitives
- MetamodelInformation.version removed
- Ini-file style information written into MetamodelInformation element annotation
- TaskEdges are now derived from TaskElement

- /isOfType is moved into TaskExecutionNode and TaskDefinition
- Packages PrimitiveTypes and Classes created in EMODECommons
- Classifications connected to RDF, Trafos
- Derived AUIComponent from EMODEClassifier and PatternParticipationElement, and removed direkt derivation from EMODEElement
- Removed UMLComponent as generalization of AUIComponent
- AUIComponent.InternalComponentsRelation set to multiplicity 0..1
- AUIComponent.InteractorAttributes removed
- AUIComponentProperty was introduced
- New Packages in EMODECommons "Classes" and "EMODEPrimitives"
- New diagram in the DialogueSpace Package "AUIRefinement"
- New association "AUIComponentRefined" and new class "AUIBox""
- Set AUIComponent to conrete classes

6.9.1 Refactorings:

- ExecutableTaskNode to TaskExecutionNode
- TaskExecution to TaskBehaviorExecution
- GroupableTaskNode is merged into TaskNode
- ExecutableTaskNodeGroup to TaskNodeGroup
- Service to FCA
- ServiceMethod to FCAMethod
- ServiceParameter/Result to FCAMethodParameter/Result
- Associations ParametersOfServiceMethod, ResultsOfServiceMethod, ServiceMethods, FCACallDefinition with their ends
- Recreated diagram FCAService as FCAWithMethods

6.10 Changes to version 1.151:

- Constraint "EMODEParameterGroup_4" was created
- DIR*INOUT is removed (and refs deleted in ParamOverloadableElement*1,)
- Parameters defaultValue and ParamDirection are moved from EMODEParameter into ParamTypeElement
- Consequently ConceptNode looses its ParamDirection Attribute and a constraint is introduced that sets ConceptNode. ParamDirection to readonly, derived
- inheritance from ConceptedElement is moved from EMODEParameter to ParamTypeElement
- ParameterOfGroup is made bidirectional
- The EMODEParameter is removed
- EMODEVisibilityKind was created and all references moved (EMODENamedElement)
- Root and Leaf attributes were removed from the enumeration EMODEAggregationKind
- The associations ServiceMethods, ParametersOfFCACall, ResultsOfFCACall, ParametersOfServiceMethod, ResultsOfServiceMethod were made bidirectional
- GroupableTaskNode is created
- /inTaskNodeGroup is moved from TaskNode to GroupableTaskNode
- TaskNode_1 (an ExecutableTaskNode may only be owned by one TaskNodeGroup) is moved to GroupableTaskNode, too
- containedNode attributes of ExecutableTaskNodeGroup is changed to type GroupableTaskNode [0..*]

- Constraint TaskObjectEdge_2 was enriched to allow type expansion (a type to one of its components)
- Attribute AUIInteractor.NeededModalityRestrictions [1..*] moves to AUIComponent.ModalityRequirements [0..*]*
- AUIInteractor.InteractorAttributes and AUIComponent.RefinedProperties removed in favor of AUIComponent.InteractorAttributes [0..*]"
- EMODEInteractionFlag removed and element usages deleted in:
  - TaskEdges
  - ExecutableTaskNodes
- TaskRealization, TaskImplementation, TaskIsEnactedBy & TaskIsInteractedAt associations were made bi-directional
- Concept Element is the interface to the rest of the model
- clash of ""name"" attribute had to be fixed. Renamed name attribute in classes UniformResourceIdentifier and XSDbuiltinPrimitveType
- DefinitionIsEnactedBy association is introduced

6.11 Changes to version 1.15:

- Package "DomainConcept" replaced with OWL & RDF
- new package "Transformations"
- New PackageControl Directory for DomainConcept
- Transformation PackageControl
- all "/StatementForGraph" texts replaced by ""StatementForGraph""
- a space in an association name of the new DomainConcpet Package was removed

6.12 Änderungen auf Version 1.143:

- Verbindung "FCACallDefinition" wird beidseitig navigierbar
- Verbindung "AssociatedSubDiagram" zwischen Diagram und DiagramElement erstellt

6.13 Änderungen auf Version 1.142:

- AUIComponent wird EMODEElement, damit es zeichenbar wird (v1.142)

6.14 Änderungen auf Version 1.141:

- ExecutableTaskNodeGroup.containedEdge wurde von Typ UMLActivityEdge auf Typ TaskEdge umgestellt
- SubGoalOf Relation wird bidirektional navigierbar
- EMODEParameterSet stand im Namenskonflikt mit EMODEParameter (wegen IDL Generierung) --> Umbenennung in EMODEParameterGroup
- AUIInteractorInteractionWays wird entfernt (ungenutzt)
- ConceptNode.directionKind wird in ConceptNode.ParamDirection umbenannt; ebenso EMODEParameter.directionKind und EMODEParameterGroup.directionKind
- Neues Element "MetamodelInformation"
- Namensänderung für enumAUIInteractorInteractionDirections, enumInteractorNecessity und enumTaskNodeTypes
- In der Ownership Assoziation (EMODECommond::Generic) müssen die Assoziationsenden in ownedElements respektive owner umbenannt werden.
- 1.141 hat die Ableitung von TaskExecution entfernt

- Nameclash zw. NamedElement & UMLClass sind resolved
- AUIInteractor ist ConceptedElement

7 Appendix D: Metamodel Documentation

This appendix includes the full documentation that was built parallel, when designing the metamodel. It is comprised of the documentation which was entered when modeling in the Enterprise Architect tool.

For reasons of document length, this documentation was separated out into another document.

8 Appendix E: Partner Feedback

We received the following partner feedback regarding model-model transformations.

SAP Research, Matthias Heinrich:

```
„Also auf den Punkt gebracht, wenn alles wie gewünscht bei der Trafo
funktioniert, können sie den Modellierungsaufwand enorm verringern. Und
folglich die Leistungsfähigkeit unserer Entwicklungsumgebung enorm
steigern! Allerdings hat sich auch gezeigt, wie schwierig es aktuell ist,
stabile M2M-Trafos basierend auf QVT zu realisieren. Folgende Faktoren
haben die Entwicklung - meiner Ansicht nach - wesentlich erschwert:
- in Entwicklung befindliche QVT-Engine
- Anpassung an neue Metamodellversionen
- geringer Tool-Support zum Erstellen von QVT-Transformationsvorschriften
- wenig Erfahrung bzgl. M2M-Trafos basierend auf QVT
- verteiltes Know-How / verteilte Entwicklerteams von QVT-Engine, QVT-
Transformationsvorschriften sowie der Java-Anbindung“
```

9 Appendix F: List of Figures