EMODE

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

German EMODE Meta Model

Telecooperation Report No. 4,
The Technical Reports Series of the Telecooperation Research Division,
TU Darmstadt
ISSN 1864-0516

We thank all partners who contributed to this document. Contributors, in alphabetical order:

Alexander Behring, TU-Darmstadt
Karin Fetzer, SAP
Gerald Hübsch, TU-Dresden
Thomas Hamann, TU-Dresden
Rene Neumerkel, TU-Dresden
Andreas Petter, TU-Darmstadt
Joachim Steinmetz, TU-Darmstadt

DOCUMENT INFORMATION	
TYPE	Deliverable
ID	D2.2 German Meta Model
DUE DATE	May 31st 2006
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	**Alexander Behring, TU Darmstadt**	**24.07.2006**
WP LEADER	**SAP**	
APPROVED	**SAP, TU Dresden**	**25.07.2006**

REVISION HISTORY			
DATE (dd.mm.yyyy)	**VERSION**	**AUTHOR**	**COMMENT**
03.06.2006	0.1	Alexander Behring	Initial Draft
21.06.2006	0.15	Alexander Behring	More packages documented, integration of comments from Karin Fetzer
24.06.2006	0.2	Alexander Behring	Full document completed
24.07.2006	1.0	Alexander Behring	Integration of comments from collaborators: Andreas Petter, Gerald Hübsch, Joachim Steinmetz, Karin Fetzer, Rene Neumerkel, Thomas Hamann

AUTHORS' CONTACT INFORMATION				
NAME	**ORGANISATION**	**EMAIL**	**TEL**	**FAX**
Alexander Behring	TU Darmstadt	behring@tk.informatik.tu-darmstadt.de	06151-16-6670	-3052

Table of Contents

1 Introduction

Techniques for software development dramatically evolved in the last two decades. Model-based design was introduced and facilitated a better communication between different stakeholders. In 1997 UML was accepted as a standard by the OMG. In the following, the OMG started working on an architectural concept for model based development of software, the Model Driven Architecture (MDA). The MDA-Guide [MDAGuide03] published mid-2003 gives a detailed explanation of the utilized concepts. The main idea is to transform platform independent models into platform specific models and thereby improve the reusability.

As summarized in the project proposal [EMODE05], there are still great challenges in the field of model driven software development. One of the key issues is the need to find a consistent integration of system and user interface modelling. Also, with the Ubiquitous Computing area emerging, the topics of modelling multimodal interaction as well as context dependence have to be taken into account. This document of the EMODE consortium would like to contribute to the research in these fields and introduces a meta model for such applications.

1.1 Development of the EMODE Meta Model

The following corner stones for the development of a meta model in the context of EMODE have been identified.

- *Control over UI look and feel.*
  Myers identified in [Myers00] that unpredictability of user interfaces may pose a high threshold to developers. This holds particularly, if the user interfaces are specified once and then adapted to the respective circumstances. Therefore, the UI meta model package should ideally give the developer the needed degree of predictability and influence on the final user interface.

- *Utilize Model Driven Architecture (MDA) concepts.*
  In the brief introduction, MDA was introduced as the next step after modelling with meta models. EMODE should not necessarily blindly follow the OMG proposal for MDA [MDAGuide03], but rather use the concepts of the MDA that apply to EMODE.

- *Improved integration of UI and system development.*
  Other projects use task models[1] as a central approach for UI development. The Cameleon project [CAMELEON] for example, introduced a framework [Calvary03] and tool support [Mori03]. The USIXML initiative [USIXML] works on a XML based UI model with a task model as its key ingredient [Usixml04] and provides several editors therefore. These projects mostly take the task model as a starting point for development, deriving other models and user interface descriptions from it.
  Within the EMODE project, the approach using task models is used, too. But since a task does not only describe something the user has to accomplish, we extend it to the system, as well. The idea is to include the system and user in one central model – the task model. A task model in EMODE describes different relations between its components, which are partitioned into user, interaction and system tasks Thereby, not only the UI design is supported, but also system implementation issues are defined on a fairly concrete level, since concrete tasks, the system has to accomplish, are being defined.

[1] To be more precise, the term “task diagram” should be used, because it is a diagram which is part of a (greater) model. But since “model” is commonly used in this context, when talking about instances, this text refers to models whenever a diagram OR model is meant.

For development and analysis, separate perspectives on the task model could be generated for special editing tasks or presentation purposes. E.g., a user view could document the tasks, only the user has to contribute to. There are essentially two ways to create a different perspective, which both leverage the model based approach:
- change the notation and keep the semantics of the elements or
- transform the model into a different meta model having different semantics.
Depending on the purpose of use, both might be relevant for developers.

- *Support for Service Oriented Architecture (SOA).*
  In the proposal [EMODE05], a component based runtime environment was chosen. To fully exploit this dimension of reusability, Service Oriented Architecture was chosen in order to support this component based runtime environment.

- *Context-awareness.*
  The application should adapt to the current context. This could have effects on interaction tasks that can be started, modalities used, etc. Context-awareness could be realized through a context server, which integrates several context services into a common infrastructure. This is further detailed in deliverable D3.1.
  For modelling the possibility to define reactions on context changes and state queries to the context is important. Separate to this is the need to model how the context information is derived, which should as well be supported by the EMODE meta model.

- *Semantically close to UML*
  UML was suggested as a starting point in the proposal [EMODE05]. To leverage the experience built into UML, concepts in the EMODE meta model should be semantically close to UML concepts. This also reduces the threshold for developers familiar with UML to use the EMODE meta model.

Based on these corner stones, the meta model has been developed. Special focus was put on the task, UI and functional core adapter model integration. This meta model will in further deliverables of the EMODE project (the project is presented in [EMODE]) be used to build editors and construct a runtime environment. Finally, the meta model and the tools will be evaluated through two demonstrators.

1.2 Introductory Remarks

In order to facilitate a better comprehensibility and discussion of the concepts and the meta model introduced in this document, some introductory remarks are made here.

1.2.1 Plattform Independence

The notion of platform independence is not clear, particularly in the definition of what “independent” is. This document uses a notion that is close to the one used in the MDAGuide [MDAGuide03]; “independence is relative”, to formulate it distinctively.

The idea of “independence is relative” is best accessed through an example. A picture is displayed **independent**, whether the platform is a PDA, a PC with a beamer or a sub notebook. A textual information is delivered to the user **independent**, whether the interaction takes place through a visual or a voice platform.

So the idea is that *platform independence always abstracts from a set of constraints, i.e. a platform, but is not defined as absolute*. To put this into MDA terms: the outcome of a Platform

Independent Model (PIM) → Platform Specific Model (PSM) transformation, the PSM, can be the input, the PIM, for the next transformation step. This idea is the basis for the user interface refinement process defined in this document, section 2.4.2.3.

The definition was chosen also, because it takes into account that most applications will be designed not to support all possible modalities there are, but rather a selected subset of them. Platform independence in this case should mean to be independent of the selected subset of modalities.

2 Model Packages

The meta model introduced in this document consists of several packages. Each package addresses a certain aspect of software development in EMODE. These aspects could be development phases, as well as models of certain aspects of the system within one development phase (c.f. deliverable D2.1). In the following the different packages are introduced, some packages are presented together, since they are closely related.

2.1 About this Documentation

Throughout this document, not all elements in all packages are introduced. The purpose is rather to give the reader a structural overview and restrict this documentation to the "what and in what way" of the model. For detailed information, in the appendix, an automatically generated documentation of the full meta model is given, including the documentation comments in the meta model itself.

We follow a nomenclature throughout this document, where a package denotes a sub unit within the EMODE meta model to group meta model elements within (e.g., all task related elements are grouped into the task package). A model is in its turn an instantiation of the EMODE meta model and contains instances of meta model elements (which are from one or more packages). To denote a model that uses elements mostly of one package and aims at modelling the aspect that package was created for, we refer to the package in the name; e.g. task model for a model of almost only task elements. Finally, a diagram is the visual presentation of (parts of) a model. I.e. a diagram doesn't have to contain all elements contained in a model but rather only those that contribute to the topic addressed by the diagram.

As a basis for development of the EMODE meta model, a bugfixed and reduced UML 2.0, called EUML [EUML], has been used. The meta model in the current version can be understood as a specialization of UML. A short overview of the UML elements used (i.e. not all) is given in the appropriate sections. A detailed explanation is out of the scope of this document, but extensive literature is available (e.g., a collection of tutorials at [Jeckle], or books like [Kath03] and [Pilone05]).

2.2 Overview of the Packages

For the meta model, only part of the software development process was regarded. This was decided, since the focus of the EMODE project was set on the later phases and the problems arising there, e.g., integration of context awareness, multimodal UIs, etc. With these challenges in mind, requirements engineering and other earlier phases are left out in order to concentrate on the main challenges.

As the starting point for the meta model supported development process, modelling of goals was chosen (see section 2.3). Using the elements of the task package (section 2.4), the developer depicts how the goals should be reached. Closely related to this, the DialogueSpace package (section 2.4) enables the developer to model the User Interface (UI) in both abstract and more concrete forms.

Using the Modality package, the developer specifies restrictions on the modalities which can be used in the UI. The DomainConcept package defines elements to model what the world of the application looks like and the Context package elements connect the concepts via the context service[2] to the real world. Furthermore, with the elements of the FunctionalCoreAdapter package, services used by the EMODE application (e.g., data retrieval and storage, computations, etc.) are defined in the terms of a SOA. And finally, elements common to all packages are specified within the EMODECommons package.

2.3 Goals

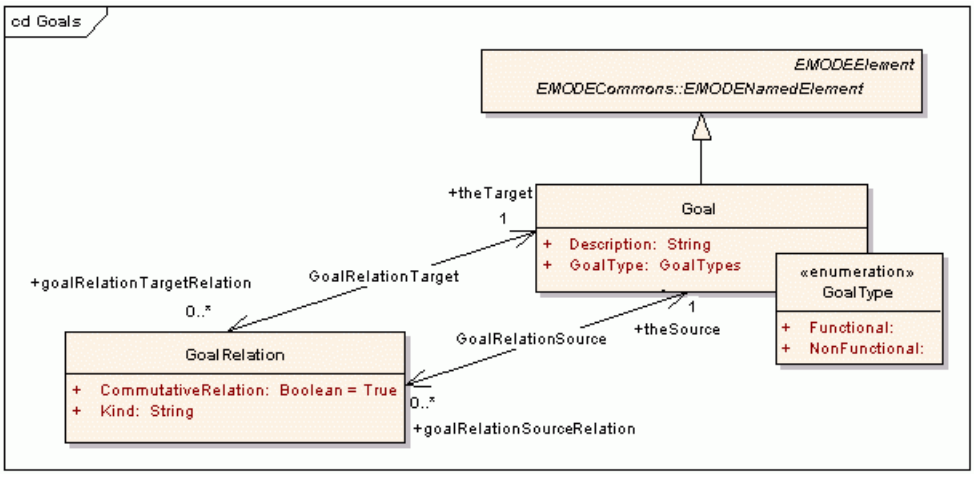


Figure 1: Goal package. Goals, their types and relations between goals are defined.

Derived through previous analysis steps, goals are formulated using **Goal** elements (see Figure 1). These goals are development aims, which the application is designed after and documented with the elements provided in the Goal Package. The formulated goals can be partitioned into (enumeration **GoalType**)

- *functional goals*,
  which describe a specific functional aspect of the targeted system (e.g., entering customer information) and therefore directly corresponds to task elements, and
- *non-functional goals*,
  which describe aspects of the target system that are not functional and therefore can not directly be mapped onto a part of the implementation (e.g., highly contrasting colors in order to allow the UI to function in bright environments). These are mostly orthogonal to functional goals and can rather be realized through “how is the model designed”.

Goals themselves can be related, which is expressed through the **GoalRelation** element. The relation is enriched by the attribute Kind, which is used to differentiate between different kinds of **GoalRelations**. This differentiation might by helpful, e.g., when performing transformations on a class of **GoalRelations**.

[2] A context service integrates different sources of contextual information and provides a common interface and possibly computation on different information sources.

2.4 Task and DialogueSpace

Closely connected to the Goal Model is the Task Model, because a task can be associated with one or more goals as it aids in reaching them (vice-versa, a goal can be connected to one or more tasks). Also essential is how a certain task is communicated to the user. Therefore, the DialogueSpace package is heavily connected to the task package.

In the first subsection, the Task Package is introduced. With it, a Task Model of the Application can be created. The Task Model specifies what tasks exist, how tasks interact and what their relations are. Hereby, the behavioural aspect of the applications and thereby the user interface is described. The second subsection introduces the DialogueSpace Package. It, in contrast to the Task Package, is used to model the layout of the user interface. Hereby a clear separation of the two aspects is accomplished.

2.4.1 Task

In the following, the Task Package is introduced. It models the behaviour of the application. Therefore, UML Activity Diagrams are introduced first, since the Task Package is a specialization thereof. The following subsection gives an overview of the adaptations made when going from UML Activity to Task Package elements. In the next subsection, 2.4.1.3, task nodes are detailed and in section 2.4.1.4 means of nesting are detailed. The fifth section describes connections between task nodes using task edges, and the following section details aspects of such connections. In subsection 2.4.1.7, a concept to reference task nodes is introduced. This section on the Task Package then concludes with a description of relations of the Task Package to other packages (except DialogueSpace, since this is detailed in section 2.4.3).

As described in the introduction, the EMODE meta model is a specialization of UML. For task modelling, this means that the task meta model package is a specialization of *UML Activity Diagrams.* This is due to the fact that Activity Diagrams cover a great deal of what we would like to express: separate activities to be performed, connection of activities with guard conditions and the possibility to pass data between them.

2.4.1.1 UML Activity Diagrams for EMODE in Brief

Activity Diagrams are made up of nodes connected with edges, all possible nodes (e.g., Final and Fork) and edges (e.g., control flow type) are derived from these base classes. A node depicts an *activity*; the edges transfer the control or data from one *activity* to another. Special nodes exist, e.g., for starting and finishing an Activity (*initial* and *final* nodes).

Like in Petri Nets, tokens are passed along the edges and an activity only starts, when all incoming edges are “satisfied”. Activity nodes that can be detailed further are called *Structured Activity Nodes*; such a node is detailed itself again in an *Activity Diagram*. Thereby, a hierarchical structure is created.

The edges between different nodes can be of different types. There are *control flow* edges which transfer “just” the control from one *activity* to another and *object flow* edges which push information from one activity into another. Edges have *guard conditions* connected to them in order to restrict when a flow is passed along the edge.

UML Input and *Output Pins* represent parts of an *Activity* that can send and receive objects through *object flows*; they are the “ports” of an *Activity*. Connected to these pins are *object flows*. In UML, an *object flow* must end in a pin, but it does not have to originate from one. For *control flows* the same characteristics hold.

By using *fork* and *join* nodes, simple Boolean operations can be performed on *control flows* in order to split up and join a set of *control flow* edges. *Decision* and *merge* nodes perform a more complex decision and merging process on *object flows*.

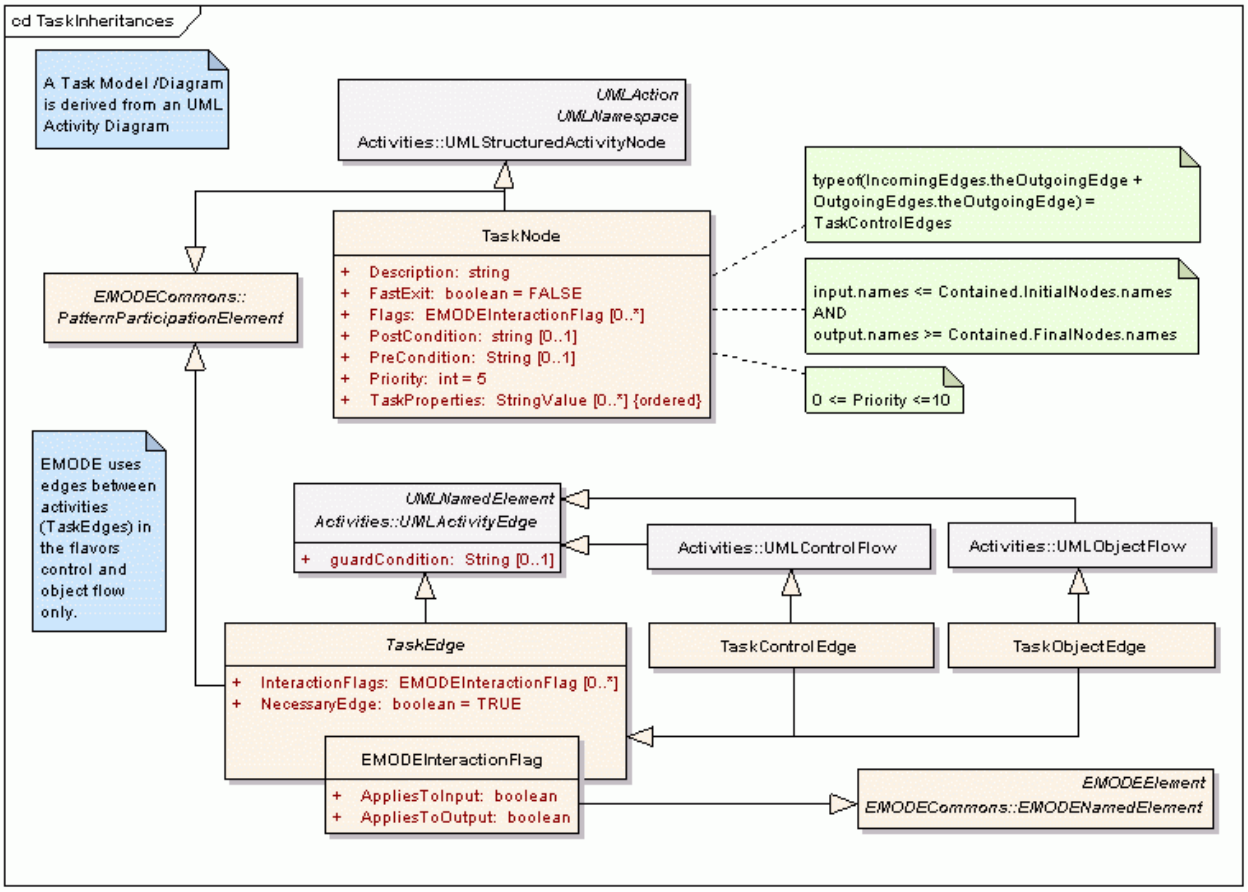


Figure 2: Task package and its inheritance from UML Activity Diagrams.

To define a starting and end point of a collection of activities, *Initial* and *Final Nodes* are needed. They are specializations of an *UML Activity* and therefore can be (just as other *Activity Nodes*) connected via *Activity Edges*. Within an *UML Activity Diagram*, they depict where a set of activities starts and ends.

2.4.1.2 From Activity Diagrams to the Task Package

The specialization of Activity Diagrams to model EMODE applications has been done in different places (see Figure 2). A **TaskNode** is introduced as a specialization of a *UML Structured Activity Node* and a **TaskEdge** as a specialization of an *UML Activity Edge*. Thereby the hierarchical structure (containment) is introduced for EMODE task models. This very useful key feature of CTT [Paterno01] is in contrast to CTT handled less strict in EMODE. CTT has a tree structure with no interconnections between branches, whereas in EMODE this latter restriction is abolished.

Input and **Output Pins**, **Initial** and **Final Nodes**, as well as **Fork, Join**, **Merge** and **Decision Nodes** are not refined further with this meta model. They convey the same meaning as described in the last section.

2.4.1.3 Task Nodes

To facilitate the development of the interaction between user and system, all **TaskNodes** drawn in a task model need to be associated to one of the partitions *user, interaction or system* (cf. **Fehler! Verweisquelle konnte nicht gefunden werden.**). This partitioning depicts in which part of the

interactive system the task is fulfilled. Hereby, the interaction partition denotes an interaction between the user and the system that takes place. The user partition denotes that the user is fulfilling the task, whereas the system partition indicates that the system must operate to fulfil it. So for example a task “decide what hotel to select” could be a user task, “generate hotel list” could be a system task and “show hotel list” could be an interaction task.

For the specialization of an *UML Structured Activity* to a **TaskNode**, several attributes were added. The Flags attribute (also given for **TaskEdges**) takes unary relations of type **EMODEInteractionFlag** that can be defined by the developer or be predefined through runtime environment support (e.g. the criticality of a task, which later results in the UI being presented on as many as possible devices). By setting the AppliesToInput and AppliesToOutput information, it can be implied where this relation operates (e.g., a message that is important to be shown to the user but the feedback is not). Naturally not all **EMODEInteractionFlag** defined can be used for both, **TaskNodes** and **TaskEdges**, since edges and nodes are of different semantics.

Some attributes for behavioural control were added. For **TaskNodes**, Pre and Postconditions can be defined through the respective attributes. If a Precondition is violated, a task should not be executed. In contrast, Postconditions describe what a task expects to produce. It thereby serves as a consistency control to be used for, e.g., throwing errors in case of vialotaion. A possible formulation of these could be in *Object Constraint Language (OCL)* [OCL2] yielding Boolean expressions or they could be realized as a reference to a Java class.

When two or more **TaskNodes** are competing for interaction resources, the priority attribute (from 0 to 10 with 10 being high priority) can help to identify the more important task. Finally, with the TaskProperties attribute, the developer can add information which is thought to be important and could be used in developer defined transformations.

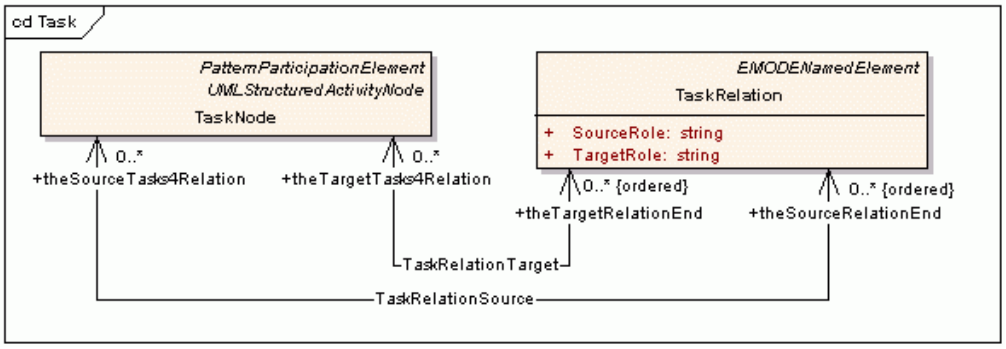


Figure 3: TaskNodes and their interrelations.

With the **TaskRelation** element, relation types between **TaskNodes** can be defined by the developer. For example “*suspend*”: a set of relations blocking the execution of another set of relations. In contrast to the Flags attribute in the **TaskNode** element (which is an unary relation), this is an n-ary relation. A special relation (an ordering relation) between task nodes is presented in the next section.

2.4.1.4 Task Nesting

Through derivation of **TaskNodes** from *UML Structures Activity Nodes*, **TaskNodes** can be nested, as mentioned in the last section. This nesting is accompanied with the question: how are the elements in the diagram detailing the super task related to the super task? This question is answered in EMODE by specifying two characteristics:

- how the detailing diagram and its elements are associated to the super task, and
- how attributes and associations of the super task are related to the sub elements.

The first aspect is addressed through the derivation from an *UML Structures Activity Node*. The UML definition takes care that the detailing elements are collected in the *ownedActivityEdges* and *ownedActivityNodes* attributes of the *Activity Node*.

All associations and attributes applied to the super task as a whole and are not distributed in any way over its sub tasks. The only attribute of a **TaskNode** affecting the interplay with its sub components is the FastExit attribute, which only affects the tasks termination behaviour and will be investigated later. The same holds for associations that are connected to a task, they do not influence its internal structure and internal behaviour. The only element that connects the task node and its sub components are the **Input** and **Output Pins** of the task.

Input and **Output Pins** constitute an interface of the super task node and therefore needs to be implemented by its subcomponents. Similar as with interfaces in Object Oriented Programming (OOP), the implementation needs to implement all elements of the provided interface. Literally, this is means for **TaskNodes** that for every **Input Pin**, there needs to be an **Initial Node** defined in the diagram detailing the task – there may be more, though. For the **Final Nodes** and **Output Pins**, this holds in reverse. I.e. for every **Final Node** there must be an **Output Pin**, but there could be more **Output Pins** than that. The identification of the **Input** and **Output Pins** with the **Initial** and **Final Nodes** is done through their names: they must be equal to produce a match.

If the **TaskNode** is connected to a **FCACall**, no sub tasks are defined within and therefore these restrictions do not apply. The same holds for User Tasks, as they are not detailed further, too. Furthermore, User Tasks cannot process objects but rather be started only. It follows that **TaskObjectEdges** cannot be connected to them.

2.4.1.5 Task Edges

TaskEdges are partitioned into **TaskControlEdges** and **TaskObjectEdges** (from *UML Control* and *Object Flows*). **TaskObjectEdges** transport data from one node to another and can only enter a **TaskNode** through an **Input Pin** (small boxes attached to the **TaskNodes** in **Fehler! Verweisquelle konnte nicht gefunden werden.**). **TaskControlEdges** depict the control flow and can be connected to an **Input Pin** or the **TaskNode** itself.

The specialization of an *UML Activity Edge* into a **TaskEdge** is refined by further giving the attribute NecessaryEdge. Hereby, the developer can define whether an edge must be “active” in order to start the subsequent **TaskNode**. For **Fork** and **Join Nodes**, this is not important, since – if the edge is not active – it may just be defined as false in the Boolean expression.

2.4.1.6 Connecting Task Edges to Task Nodes

In the following the semantics of starting a task and task termination is detailed. In contrast to *UML Activity Diagrams*, Tasks Diagrams incorporate the idea of Petri Nets at a reduced level, only. This means that incoming edges may be optional and not required to start the task. As well, not all outgoing edges will have a token when a task finishes.

Two different intentions may be present when connecting an incoming task edge to a task:

- use this edge only to influence whether a task is started or not, or alternatively
- affect the processing of the task and pass information to it.

The difference of these two can be compared with programming: the first option is whether you call a function, the second means calling the function with parameters.

The first of the two options is equivalent to a **TaskControlEdge** that should "just" start the task. Such edges can hence be connected to the **TaskNode** itself, because they need no internal representation through an **Initial Node**. It does not matter whether the **TaskNode** is of type User, Interaction or System, in this respect, the semantic is the same for all of them.

For the second option, the task edge should influence the processing of the task's sub tasks or the behaviour of the associated **FCACall**. The task edges therefore need to be connected to the "interface" of the task, the **Input** and **Output Pins**. If the task is structured, the contained **Initial Nodes** will be active if the corresponding **Input Pins'** task edges are.

For output, again, similar semantics apply. On task termination, all **TaskControlEdges** connected directly to the **TaskNode** are set active. Task edges connected to **Output Pins** are set active if the connected **FCACall** sets them so or the corresponding **Final Nodes** in the structured task were active when the task terminated. But here, the question arises, when is a task terminated? The FastExit attribute (of the **TaskNode**) is introduced to distinguish two termination behaviours.

If FastExit set to true, all sub tasks inside a structured task are terminated immediately when any **Final Node** goes active. Otherwise, with FastExit set to false, termination waits until termination of all sub tasks, active the moment the first **Final Node** goes active. For termination the output of those sub tasks is taken into account when they are connected to **Final Nodes**. This way, the output of all "last" sub tasks is considered, and the **Output Pins** of the super task set accordingly.

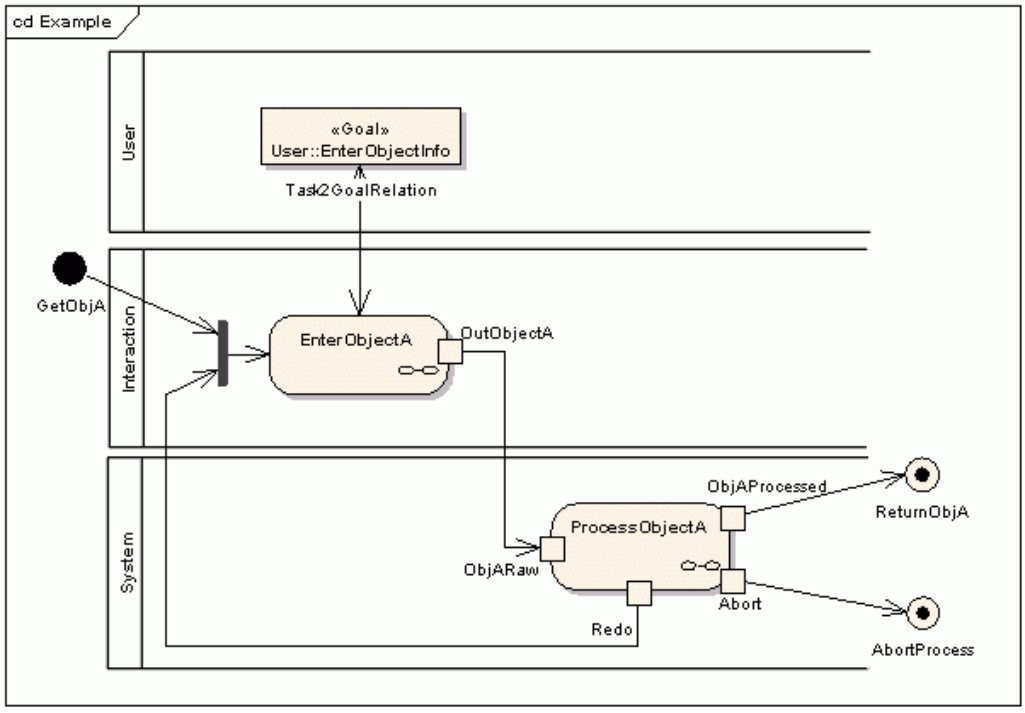


Figure 4: A task model example. Refer to the text for a detailed explanation.

TaskNode and **TaskEdge** usage is exemplified in **Fehler! Verweisquelle konnte nicht gefunden werden.**. Depicted is a task diagram, which shows part of a task model. The task diagram is the detailed specification of a super task called "Example". Looking at **Initial** and **Final Node**, the super task maximally has one **Input Pin** (*GetObjA*) and minimally the **Output Pins** *ReturnObjA* and *AbortProcess* (c.f. section 2.4.1.4).

Through the subtask *EnterObjectA*, the functional goal *EnterObjectInfo* is reached, the goal is, as it is drawn in the User Partition a goal of the user. The task *EnterObjectA* connects a **TaskObjectEdge** through the Output Pin *OutObjectA*. This data is being processed by the system in task *ProcessObjectA*. The latter task can activate two different **TaskObjectEdges** (*ObjAProcessed* and *Abort*) and one **TaskControlEdge** (*Redo*). No statement is made whether *ProcessObjectA* can activate all outputs at the same time, just one or a combination of them.

2.4.1.7 TaskNode References

After designing a task and detailing it with sub tasks in possibly more than one level, the question how this task can be used in different situations, i.e. in multiple positions, of the model. For example, a save task that takes care of storing some information might be used in several other tasks as a final subtask. Including, e.g., the save task, does not mean that for every appearance, the task's attributes and associations are exactly the same. Therefore, only the internal nodes and edges (the "realization") may be referenced, but not the node itself, its pins or attributes.

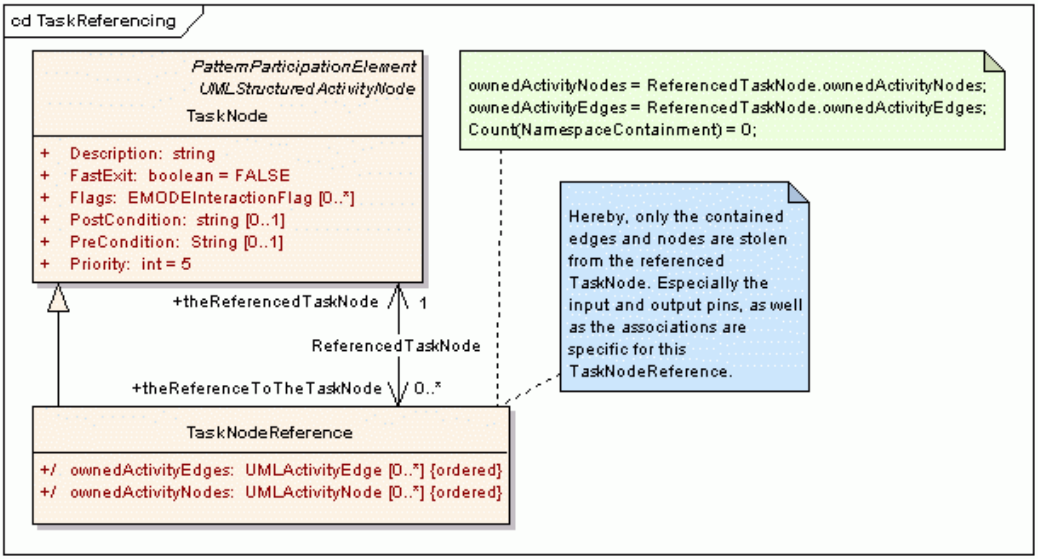


Figure 5: referencing task nodes only inherits the referenced node internals, not the node, its attributes or pins.

To accomplish this, the separation shown in Figure 5 is introduced. A **TaskNodeReference** is derived from **TaskNode**. Its contained sub edges and sub tasks (*ownedActivityEdges* and *ownedActivityNode*) are consequently declared as derived. The information for the derivation (the sub tasks and sub edges) is drawn from the referenced node by connecting the **TaskNodeReference** with it via the ReferencedTaskNode association. This association should be read from the referencing to the referenced node, but it is bi-directional, since the model should keep track of references to a node, e.g., to determine whether it may be deleted.

2.4.1.8 Relations to Other Packages

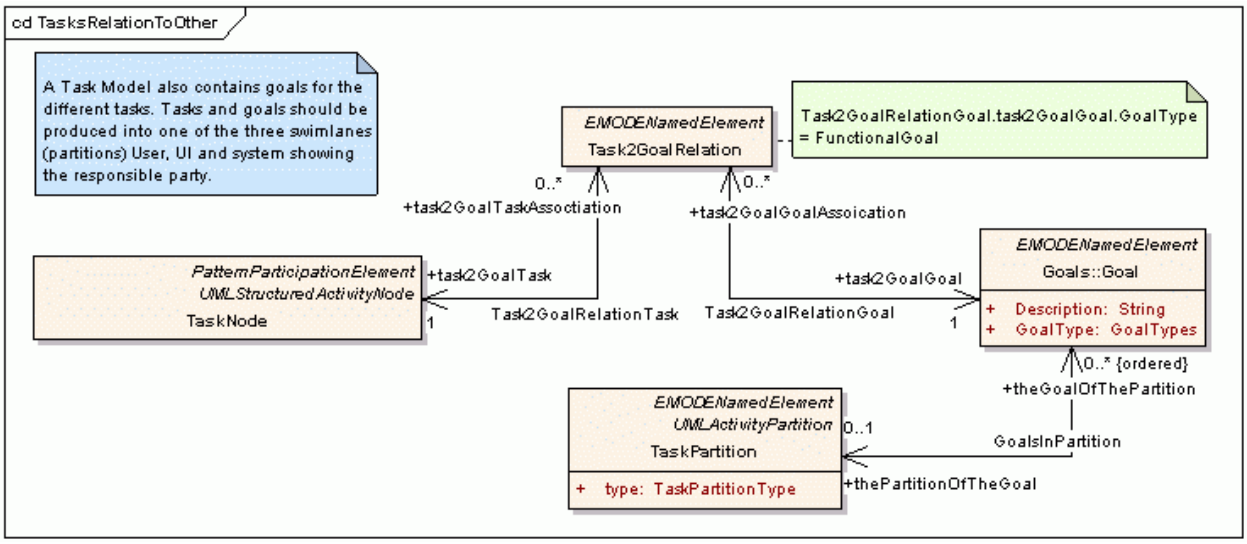


Figure 6: Task Package. Relations between tasks, partitions and goals.

In Figure 6, the relation between **TaskNodes** and goals through the **Task2GoalRelation** element is depicted: a functional goal can be associated to a **TaskNode**, meaning that the task aids in fulfilling it. The **Goals** can be grouped into the partitions *System*, *User* and *Interaction*, as the tasks can (see section 0). Putting a goal in a certain partition denotes who is striving to reach it. This information (the **Goal** itself, its placement in a partition and its associations to **TaskNodes**) can be used to, e.g., to produce hints for the runtime, like a goal based user-guidance. It as well helps in making design decisions transparent throughout the development process. However, it is not necessary to place the (via **Task2GoalRelation**) associated **Goals** and **TaskNodes** in the same partition. I.e. there may be more than one party pursuing a Goal through a certain task.

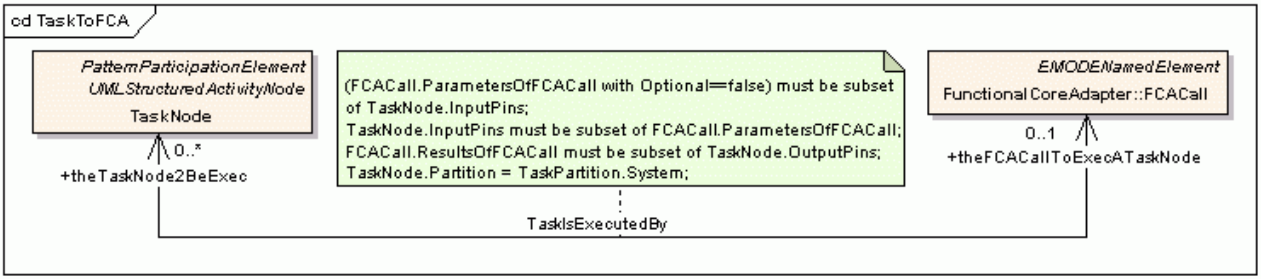


Figure 7: Association of TaskNodes with Functional Core Adapter Calls (FCACalls)

In Figure 7, the relation between a functional core adapter (FCA) call and a **TaskNode** is shown. If a task is a system task, it can be realized by calling an FCA via the TaskIsExecutedBy association. This **FCACall** in turn is (as defined in 2.7) either a call to an external or internal service. Identification of **Input** and **Output Pins** is (like when structuring tasks) done via the names. The pin names must match the names of the **FCACall's** parameters and results. A more detailed description can be found in section 2.7.

After introducing the DialogSpace Package in the next section, the connection between DialogueSpace and Task Package will be detailed in section 2.4.3.

2.4.2 DialogueSpace

This section introduces the DialogueSpace Package. It essentially facilitates modelling of the user interface. First, a distinction between DialogueSpace, abstract user interfaces as a part of it and

tasks is made. Subsequently, the elements of the abstract user interface are introduced and their general properties detailed. In subsection 2.4.2.3, the process that enables EMODE applications to cover a wide range of contexts for user interface adaptation is introduced. This section closes with a description of how relations between abstract user interface interactors are specified.

The section following this one, 2.4.3, covers a very important aspect of the DialogueSpace – its connection to the Task Package.

2.4.2.1 Distinction Between DialogueSpace, Abstract User Interface and Task Package

The DialogueSpace package is defined in the ITEA D2.2 document as "*A dialogue space is the unit construct to define AUI's. It is an abstract structuring unit that supports a set of logically-connected tasks*". *It hereby includes (partially) ordering through the tasks*." This is based on the notion of platform independence used in the ITEA D2.2. Absolute platform independence is assumed in the task model versus a platform dependent view is assumed in the DialogueSpace model.

For the meta model presented in this document, the notion of platform independence was introduced in section 1.2.1. It differs from the one above, introduced in the ITEA D2.2, and per definition does not make a statement about absolute platform independence. Therefore, a different separation between the DialogueSpace and Task Models has to be made, since the distinction platform independent versus platform dependent does not work for this meta model.

The separation of Task and DialogueSpace in this meta model is accomplished by separating definition of layout versus definition of order (one might say "semantic ordering"). I.e. in the Task Model, the order of things is documented (e.g., task A needs to be accomplished before task B), whereas in the DialogueSpace model, the layout[3] and structure of things is defined (e.g., the ok button is 20pt wide and colored green). Through this distinction, several things are achieved.

- A clear role separation what is modelled with what package,
- more compact meta models, since there are no elements conveying similar semantics in both packages, and
- support for fain grained transformations that introduce more and more platform dependence, like suggested in the MDA Guide (cf. section 1.2.1), which are leveraged to gain more control over the UI look and feel (cf. section on meta model development, 1.1).

Since this is a different notion than the one used in ITEA D2.2, we use the more suitable term "Abstract User Interface" (AUI) to refer to the description of the UI layout and structure.

The *Plasticity* of a user interface (as introduced in [Thevenin99]) denotes its ability to preserve usability under variation of physical system characteristics and environment. Furthermore, plastic UIs are specified only once.

Modelling of user interfaces in EMODE aims at increasing the plasticity of the modelled UIs, while at the same time preserving predictability. This directly addresses Myers concern in [Myers00] that one of the main reasons for the high threshold of plastic UIs still present for many programmers may be the unpredictability of the adaptation process.

In EMODE, an UI is specified once, through an abstract description; an AUI. It then is refined to the targeted modalities. EMODE breaks with the strictness of modelling a plastic UI only once and hereby addressing Myer's predictability concern. EMODE rather permits for the developer to

[3] With layout, also the temporal layout in a voice application is meant (e.g., what is interacted first)

intervene in the adaptation process. The AUI description and the AUI adaptation process will be the topic of the next two sections.

2.4.2.2 The Elements of an AUI

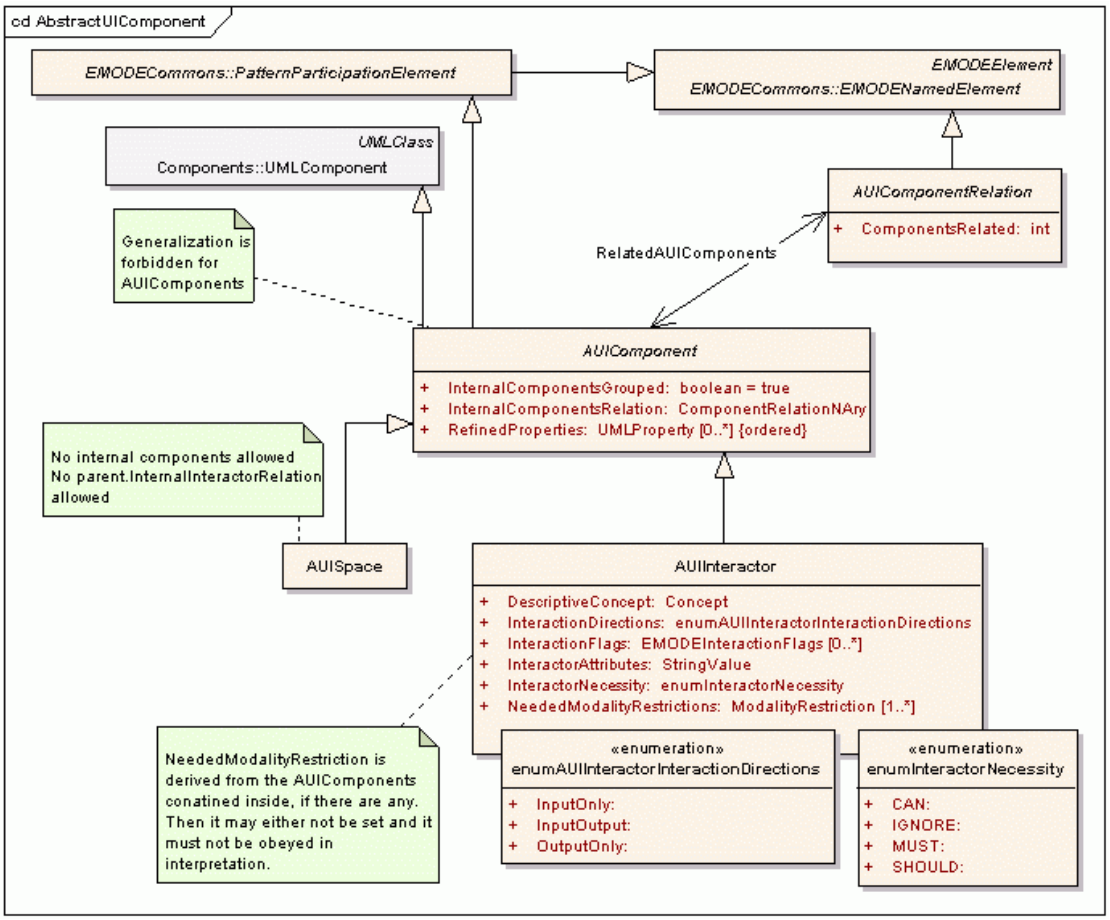


Figure 8: DialogueSpace package. The inheritance of AUIComponents.

An AUI is described in terms of **AUIComponents**, which behave according to the composite pattern; i.e., they can be nested. This is similar to major UI implementations (like Windows, SWING, etc.), where the composite pattern also is implemented. In contrast to the definition of the pattern in [Gamma95], there is no distinction made between leaf and composite nodes here. **AUIComponents** are always seen as full entities. The realization of the composite pattern is achieved by deriving the **AUIComponent** from *UML Component* and therefore being able to nest **AUIComponents**. Optionally, via the InternalInteractorRelation attribute, the developer can defined how the components grouped into a common parent component relate to each other (see section 2.4.2.4 for details).

The **AUIComponent** is partitioned into **AUISpace** and **AUIInteractor**. The **AUIInteractor** represents the elements that interact with the user (the actual abstract widget element). In contrast, the **AUISpaces** can be used to as a placeholder for an **AUIInteractor**. For example when creating a wizard like screen flow, the different interactors representing the screens could go into an **AUISpace** reserving the space for the different wizard pages. This is detailed in section 2.4.3, when the connection between Task and DialogueSpace is examined.

Generalization for **AUIComponents** is similar to generalization in class hierarchies. The specialized classes incorporate a more special behaviour. For example an m-of-n selection

component could be specialized into a 1-of-n selection component. The latter is a specialization, because it incorporates the same behaviour with more restrictions (only 1 and not m to select).

It is important to note that with these model elements, user interfaces from a very abstract level (little constrained on a specific platform) to a very concrete level (close to the user interface actually displayed) can be modelled. The meta model is designed so that all the different levels of abstraction can be modelled with the same meta model. This is beneficial for the refinement process that adapts the user interfaces to a specific context – which is the topic of the next section.

2.4.2.3 DialogueSpaces are Adapted to Context

AUIs (as described before – the AUI is the part of the DialogueSpace used in this meta model) participate in a stepwise refinement process. Through this process, an abstract description is refined to a concrete description of the user interface. The refinement hereby is dependent on the context, particularly the modality configuration. This refinement happens within the same meta model, therefore, among other things, making the process flexible in the number of refinement steps. This is in direct correspondence with the notion of platform independence introduced in section 1.2.1: every refinement step introduces a bit more platform dependence – one gradually refines the user interface. Hereby, the AUI is gradually adapted to the context.

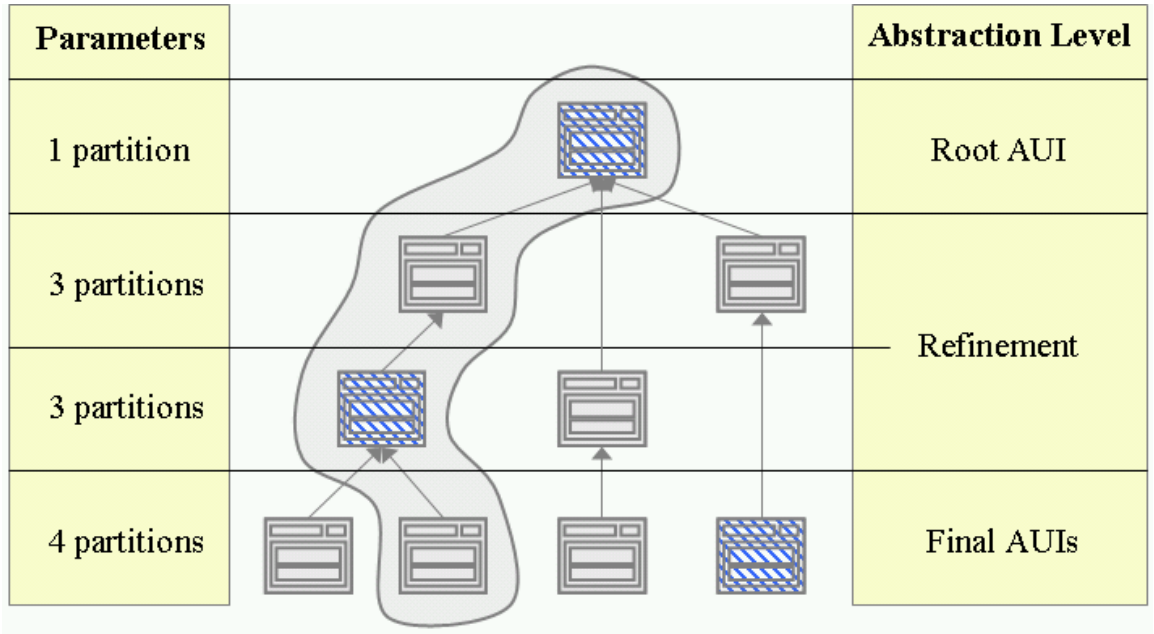


Figure 9: AUI refinement process. The boxes depict AUIs with its contained components at different levels of abstraction; striped boxes are defined by the developer.

An example of the refinement process is depicted in Figure 9. For a given user interface, depicted at the top, one “version” exists at the most abstract level. This most abstract AUI, the whole refinement process starts with, is called *Root AUI*. Three refinement steps are shown in Figure 9.

1. The Root AUI is refined for three different platforms (c.f. section 1.2.1 on platform independence), e.g., UI toolkits like SWING, XIML and GTK.
2. For two of the refined versions, a transformation is performed to, e.g., set some defaults for the UI toolkits.
3. The last refinement step refines one of the AUIs into two different versions (e.g., screen resolutions) and transforms one of the AUIs.

After these three steps, the AUIs of the example in Figure 9 have been refined to the version that should be presented to the user. This last AUI is at the most concrete level of refinement. Since

these most concrete AUIs are still represented using the AUI meta model package, they are called *Final AUIs*.

In order to not only rely on automatic transformations for this process, arbitrary AUIs in the refinement tree can be defined by the developer. For example in Figure 9, the developer defined three AUIs, depicted striped in the figure. The first one, the Root AUI[4], is at the most abstract level and it influences the UI for all possible contexts. The one on the second lowest level is the definition of an intermediate step in the refinement process. The definitions made herein are valid for a certain context and influence two Final AUIs. The Final AUI defined will be displayed to the user.

It is important to note a consequence of the process introduced here: the AUIs defined explicitly by the developer are subject to a trade-off. Either the defined AUI is valid in a bigger context (it is further up in the refinement tree) or the AUI defined is more concrete and therefore not subject to further adaptations that change it (more to the leafs of the refinement tree). Enabling this trade-off gives the developer the chance to find the right level of abstraction to formulate the user interface(s), and if necessary some UIs can be defined on the most concrete level – the Final AUI.

With the marked area in Figure 9, a possible instance of such a refinement process is sketched. The adaptation process can identify the most concrete AUI defined for the current context, which is the one on the second lowest level. Therefore the first two refinement steps can be omitted. For the chosen AUI, further refinement takes place (one step in this example) to produce the Final AUI for the current context.

From the Final AUI derived through the refinement process, a *Concrete UI (CUI)* has to be produced. The biggest difference of AUI and CUI lies in their meta models: the AUI is described with the DialogueSpace package of the EMODE meta model, whereas the CUI is described in the meta model (language) of the targeted renderer (e.g., HTML, XIML, etc.). Of course, between AUI and CUI a transformation needs to take place. Through this, further information could be added – so that the CUI is even more concrete than the AUI. But this is highly dependent on the actual implementation of the runtime environment and therefore here not specified further.

This refinement process exemplified above can take place during design and/or runtime, the UI hereby is adapted to a specific context, e.g., the modality configuration. Consequently, branches occur at every place where a parameter range (a part of the context) is split up (into two different contexts) and two different AUIs are defined for each parameter range. The refinement (i.e. how the splitting is realized) is facilitated by transformations.

This refinement of AUIs could be initiated and/or influenced by different factors, e.g.:

- by automatic generation at design and/or runtime,
- by the developer through modelling specific AUIs within the tree or defining as well as invoking transformations or
- by the user through implicitly or explicitly defining or invoking transformations, and
- any combination of the above, also with partially automation (e.g., the developer gives partial hints at development time and the rest of the needed information is drawn from the current context).

A more detailed discussion of which options are supported within the EMODE project can be found in deliverables D2.3 and D3.1.

[4] The Root AUI could also be, of course, be generated through a transformation from the Task Model.

2.4.2.4 AUIInteractor Relations

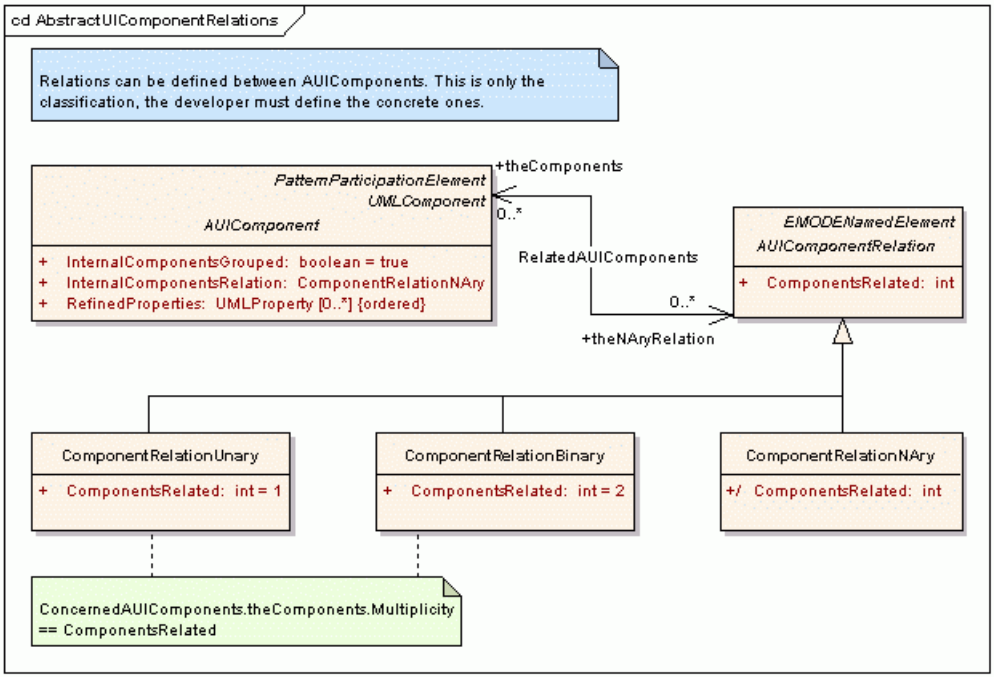


Figure 10: AUIRelations. Relations between AUIComponents can be defined.

Relations between interactors are useful to define the layout part of the user interface (cf. separation Task versus DialogueSpace Model). For example, the developer can define an ordering relation and use it to order interactors in the UI.

Relations can be specified with the successors of the **AUIComponentRelation** element. In Figure 10, the classification of a relation between **AUIComponents** into unary, binary and nary relations is defined. As introduced in section 2.4.2.2, nary relations can be used as attributes for AUIComponents. Hereby, all interactors contained in the component are related via the specified nary relation.

2.4.3 Interplay of Task And DialogueSpace Packages

There is a strong relation between the Task Package and the DialogueSpace Package. In the first part of this section, the associations, the two packages are connected with, are introduced. The second subsection describes consequences of the associations introduced in the first section.

2.4.3.1 Connecting Tasks and Interactors

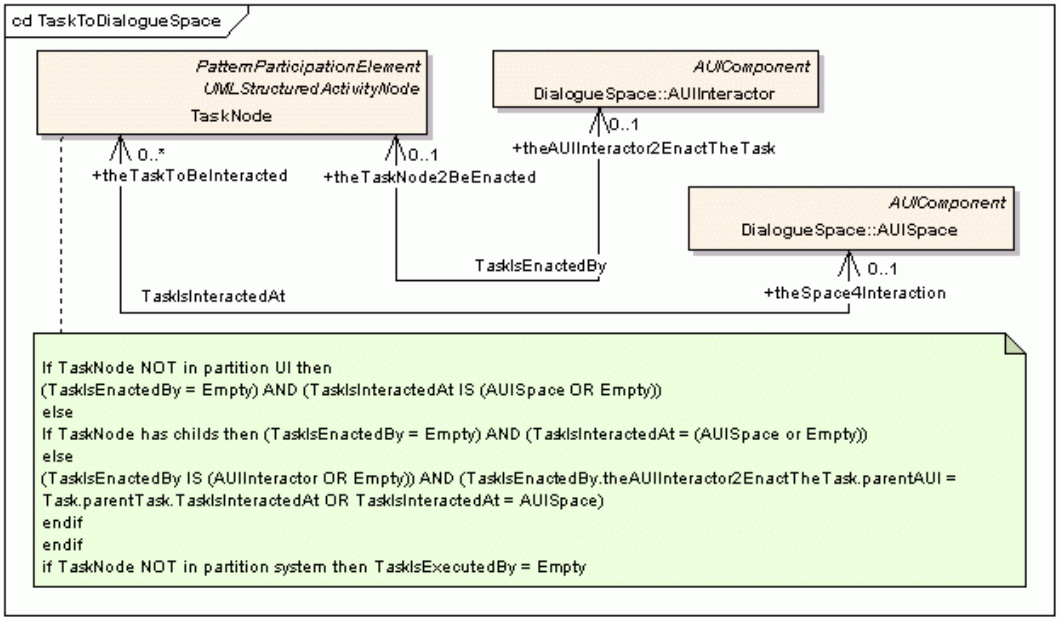


Figure 11: Connections between Task and DialogueSpace Packages.

Interaction tasks (cf. 2.4.1) need an AUI to interact with the user. For the interaction, two properties need to be defined: how and where the interaction takes place In other words, an interactor (defining how to interact) and a space (that the interaction takes place at) in the user interface are needed. Therefore, the associations TaskIsInteractedAt and TaskIsEnactedBy are introduced to connect **TaskNodes** with **AUIComponents** (cf. Figure 11). Accordingly the **AUISpace** (as a placeholder) is connected to TaskIsInteractedAt and the **AUIInteractor** (as the interactor) to TaskIsEnactedBy (cf. section on AUI elements, 2.4.2.2).

These associations can be read from Task to DialogueSpace Packages or vice-versa. In EMODE, it is suggested to see the task model as a more abstract entity and therefore to read the associations as references from the DialogueSpace to the Task Package.

2.4.3.2 Similarity in Structure of Task and DialogueSpace Packages

Since EMODE developers should have strong control over the UI look and feel, they should be able to influence the layout of the whole application. That means in consequence, it is insufficient to only have control over the layout of single interactors. The grouping of interactors, the layout of containers, interactors are grouped in, and other aspects of the application layout must be in reach to be changed by the developer.

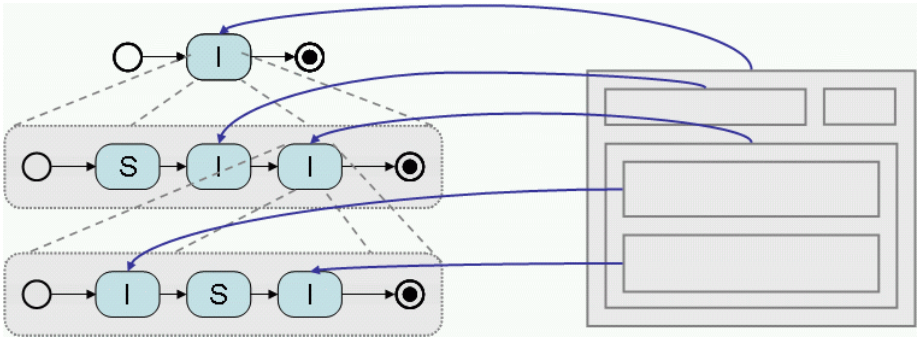


Figure 12: Task and AUI structure are similar. Depicted is a task hierarchy and an AUI with its components.

On the other hand, information from the Task Model should be used in the AUI, because EMODE situates the tasks at a higher level of abstraction than the AUIs. As for example in [Mori03], the task structure can be used to generate an abstract UI representation. The same approach is followed in EMODE, meaning that tasks hereby imply a structuring and grouping of the interactors. Hereby, statements about tasks, for example their nesting and their attributes, are reused in the AUI. Especially the nesting of the tasks hereby implies the structure of the AUI. This situation is depicted in Figure 12. The super task is mapped to the basic **AUIComponent**. Its sub tasks are executed in **AUIComponents** contained in this basic **AUIComponent**. This applies recursively to the third task hierarchy level, as well.

For example a task to edit a customer with several subtasks for editing contacts, addresses, loading, saving, etc. will have an **AUIComponent** to group the interactors for the subtasks. Through the relations between the subtasks, it can be derived, which interactors must be used sequentially and which should be able to be used in parallel, etc. The approach to derive the AUI from tasks hence facilitates reuse of information given in the Task Model for the AUI.

If the idea that task and aui structure are similar is followed consequently, AUIComponents for more abstract tasks (as the top most in Figure 12) will be created. This can be leveraged to give the developer control over the look and feel of the whole application. In the AUIs connected to more abstract tasks, the grouping and interplay of the sub interactors can be defined and therefore the layout of the whole application is adapted to the developer's demands.

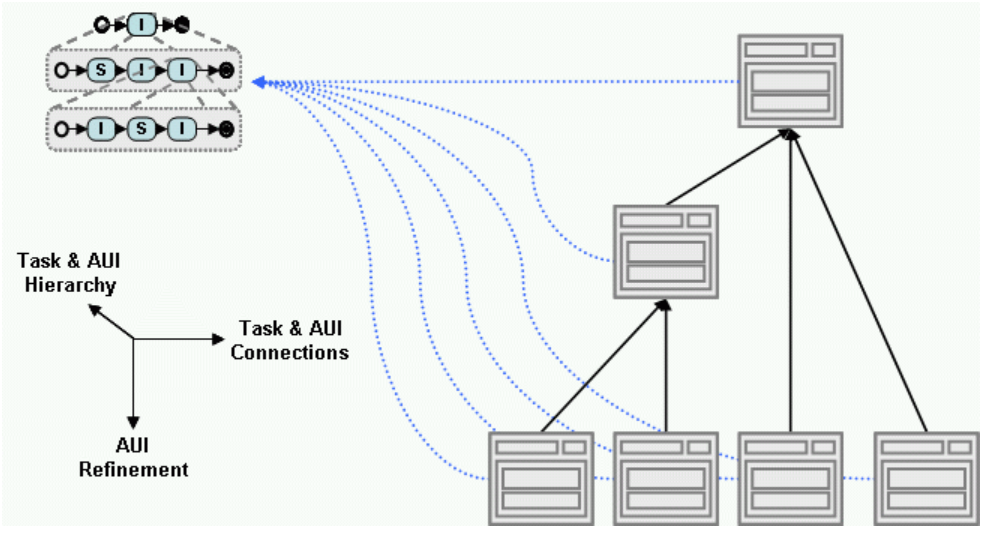


Figure 13: Task hierarchy, AUI structure and AUI refinement are related.

The relations between the task hierarchy, the AUI structure and the AUI refinement process are illustrated in Figure 13. The task hierarchy can be seen on the upper left. It shows the nesting of tasks (in the third dimension). To the right of the task hierarchy, AUIs are connected to it. Beginning with the topmost AUI, the refined versions (cf. section 2.4.2.3) are shown going downwards. These refined versions are all connected to the same task hierarchy as the topmost AUI is, through the associations introduced in the last section.

With the structure introduced, the **AUIComponents** in the Final AUIs can be traced to what task they realize. This is facilitated by the refinement process that either keeps traces about the refinement of every **AUIComponent** or by also refining the task-aui-relations. Hereby a key characteristic of the task-aui-relation is fulfilled. Relations between the components of a concrete, modality specific implementation of the user interface (Final AUI) and the tasks, which describe the processing of this information exchanged, can be identified. Pragmatically summarized: interaction (the UI shown to the user) and information processing (the Task Model) are connected.

2.4.3.3 More details on Task – Interactor Connections

As a consequence of the structure similarity introduced in the last section, **AUIComponent** referenced through TaskIsInteractedAt must be within the AUI of the parent task. For example, the **AUIComponent** interacting the sub task “edit customer address” must be placed in the **AUIComponent** interacting its supertask “edit customer”. That is more formally expressed: $T_1 \subset T_2 \Leftrightarrow AUI(T_1) \subset AUI(T_2)$, where T stands for a task and AUI(T) is the AUI, the task T is interacted at. For AUIs and tasks alike, the semantic connected to “ $X_1 \subset X_2$ ” is that X_1 is a component in X_2. This means X_1 is a subtask of X_2 respectively X_1 is placed in X_2.

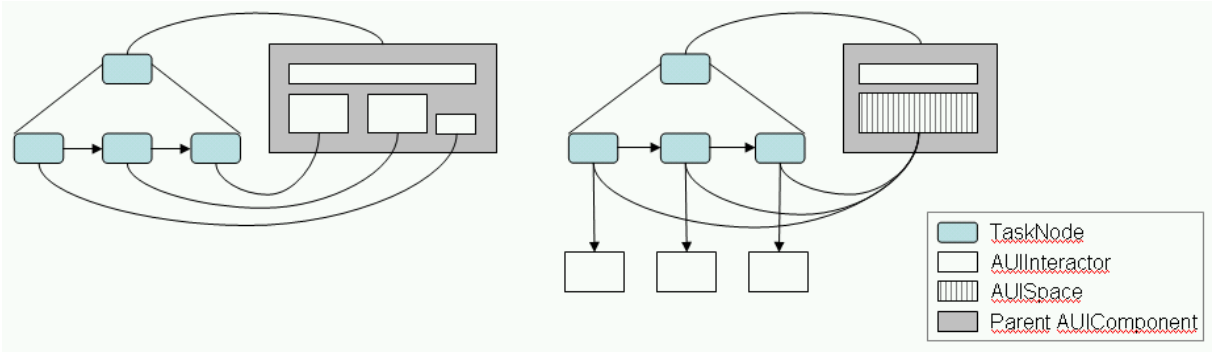


Figure 14: TaskIsInteractedAt and TaskIsEnactedBy can be different or the same.

Observing the two associations, two cases can appear in general, which are depicted in Figure 14. In the first case, the same **AUIComponent** is referenced by both, TaskIsInteractedAt and TaskIsEnactedBy. The developer hereby states that the referenced AUIComponent is the correct interactor to interact the task and is already placed in the correct position. The figure shows this situation on the left side. The sub tasks are connected to interactors that are already placed in the correct position (within the parent AUI).

The second case the referenced **AUIComponents** are different. This is depicted on the right side of the figure, where every subtask references an interactor, but separately specifies that the interaction should take place in the AUISpace depicted as a striped box in the figure.

For EMODE, a shortcut is introduced for the first case of the above situation. If the developer only defines TaskIsEnactedBy, the first case is assumed. The referenced interactor is thought to be placed correctly. This differs from the second case, where two different **AUIComponents** must

be given. Hereby the TaskIsInteractedAt component must be an **AUISpace**. It will be filled with the **AUIInteractor** referenced to by TaskIsEnactedBy.[5]

2.4.3.4 Atomic Interactions

An *atomic interaction* is the leaf interaction task together with its interactor (via TaskIsEnactedBy). It therefore comprises the behavioural aspect (the task) and the interaction layout aspect (the interactor). The abstraction level of this atomic interaction can be defined by the developer (e.g., “Select 1 of n component” versus “Windows List View”). This heavily depends on what widgets are implemented and therefore predefined for use and which widgets are made available through libraries. The interactions defined through the latter case are also seen as atomic. The Library Package is detailed in section 2.6.3.

The meta model is designed to separate the task and layout views of an atomic interaction. This separation could also be followed in the implementation. For one task implementation, several interactor implementations may be eligible. They could be combined during design and/or runtime depending on context and availability; assumed an appropriate interface has been defined. This may be at a more concrete level (for a set of tasks and interactors) or a more general level (for all tasks and interactors); c.f. [Hartl03], [Braun03] and [Calvary04]. Further details of this will be documented along with the runtime specification for EMODE.

The connection between Task and DialogueSpace Package introduced in this document has some key characteristics:

- the task and AUI structure is similar,
- a task is a more abstract element than an AUI, and
- the full look and feel of an application can be influenced by the developer.

For the design and runtime different scenarios could be envisioned. A developer could define everything in the AUI to control the look and feel. On the other hand, only the tasks could be defined and transformations derive the appropriate interactors (probably even at runtime[6]) to enable the interaction with the user. This opens up a trade-off between detailed modelling with full control over the UI versus highly efficient level task modelling with not worrying about the UI at all. There is a variety of possibilities in between that could be used to tune the development effort exactly to the problem at hand.

Of course, interactors and tasks rely in a common understanding of the communicated concepts. This is particularly the case when interactors and tasks from different libraries are introduced. This raises the need to define the concepts involved, which will be the topic of the next section.

2.5 DomainConcept and Context

This section introduces the DomainConcept and Context Packages. Whereas the DomainConcept Package facilitates the description of the concept the application uses, the Context Package is used to describe how contextual information is derived and how it is used.

Drawing from the idea of MDA to model as much as possible at a level as abstract as possible, more information than just the data types was included into the DomainConcept model. The package is based on OWL-DL and hence (among other things) relations between concepts can be specified, as well. Based on the concepts defined therein, the developer defines in the context

[5] Since an **AUISpace** can be the place of interaction for an arbitrary number of tasks, the need for validation arises. It has to be checked that a task which is still running in an **AUISpace** is not interrupted by another task demanding control of the space.

[6] Transformations at runtime might also be called adaptations

model how information (i.e. instances of domain concepts) is derived and can be used. Expressed differently, the DomainConcept model describes the application's view on the world and the Context model connects the real world to it.

Following this thought into an implementation view, a context service could instantiate DomainConcept model elements. The server hereby knows the concepts through the DomainConcept model and how to instantiate a concept by looking into the context model.

2.5.1 DomainConcept

This section details the DomainConcept Package. It is used to describe the concepts an application uses and their relations– i.e. the application's view of the world. To enable the developer to model as much abstract information as she wishes in the DomainConcept model, the decision was made to implement an ontology-like package. Additionally, this enables reasoning on concepts and their connections. The package therefore should be as close as possible to OWL DL [OWL] in order to be able to reason on it with already present tools (KAON2[7], RACER[8], FaCT[9] and FaCT++[10]). This also is an important industry concern: the OMG submitted a Request for Proposal [ODMRFP] for (among other things) creating a MOF2 compliant OWL DL meta model.

The proposals [Brockmans04], [Djuric05] and [IBM05] were considered for integration in EMODE.The criteria used when choosing one of the proposals to be used as the DomainConcept model in EMODE were:

- it should model at least OWL DL,
- it should not be too big (i.e. a full RDF implementation is too much for EMODE) and
- it should be easy to understand.

The proposal [Brockmans04] fulfils the criteria best. It contains all the needed elements and little or no overhead. The following minor changes were made in order to meet our needs better:

- The further classification of the Restriction element is not as close to OWL DL as one would expect – it is more elaborate. In order to keep the package as close to OWL DL as possible, the classification of restrictions from a different proposal [Djuric05] was incorporated.
- A match between the annotation property and developer annotations was made so that all annotations in the meta modal will be of the same type
- ConceptInstances, i.e. individuals and DataValues, can be used to define variables and objects passed between, e.g., tasks. Therefore, a **ConceptInstance** element was defined.

The graphical elements used for modelling (via an UML profile) presented in [Brockmann04] will be changed to EMODE specific representations, in order to integrate into the EMODE modelling environment.

By using the primitive types recommended for OWL by the W3C [W3C04] also the concrete representation of the derived objects is defined.

2.5.2 Context

The connection between the application world, described in the DomainConcept model, and the real world is modelled with the Context model. The purpose of the context model is to be able to

[7] http://kaon2.semanticweb.org/
[8] http://www.racer-systems.com/de/index.phtml
[9] http://www.cs.man.ac.uk/~horrocks/FaCT/
[10] http://owl.man.ac.uk/factplusplus/

formulate queries to the current application context (which is an instance of the DomainConcept model) and to define how the information in the DomainConcept model can be derived. This is done bearing in mind that adaptivity is a key criterion for the design of the EMODE meta model.

The Context Package presented here is closely coupled with D3.1, which details the context service. Particularly in this package, the interplay between meta model, runtime and design time environment is very tight. This might necessitate changes, which will be reflected in future versions of this document.

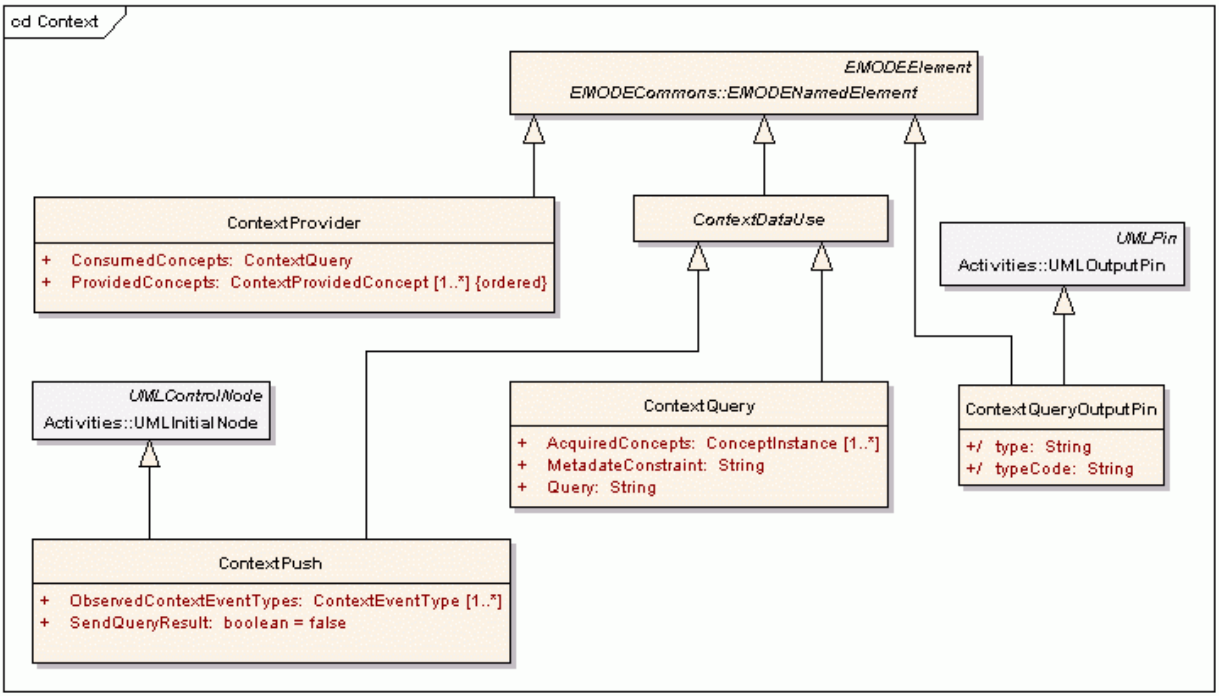


Figure 15: overview of the major elements of the Context package.

An overview of the context elements is given in Figure 15. Queries to the current context can be defined by using the **ContextQuery** element. The **ContextProvider** defines how concepts can be provided. To interface the context with the task model, a **ContextPush** element can be defined in order to start a task upon a context event. The **ContextQueryOutputPin** can be used to input contextual information into tasks via a **TaskObjectEdge** or pass the control to a task via the **TaskControlEdge** (cf. section 2.4.1)

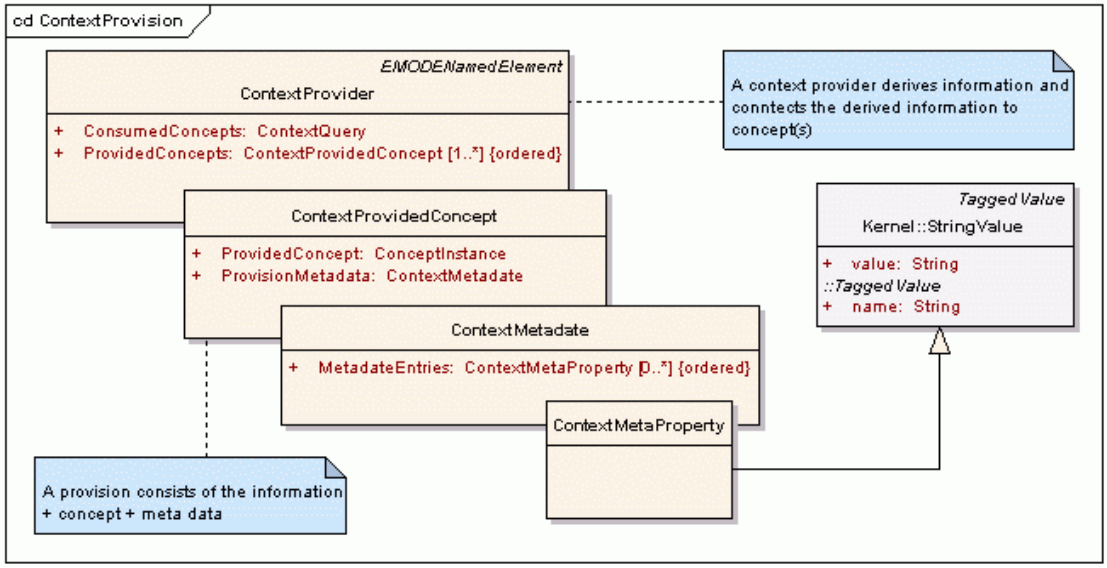


Figure 16: context provision diagram.

A more detailed picture of the context provision can be seen in Figure 16. A **ContextProvider** defines what concepts it consumes. This is done by using the **ContextQuery** element in the ComsumedConcepts attribute. From the consumed concepts, the provider derives the conepts it provides. The concepts provided are defined in the ProvidedConcepts attribute. The provision of a concept thus is detailed by the **ContextProvidedConcept** element.

A **ContextProvidedConcept** in turn carries the concept provided (ProvidedConcept) and meta-data associated to it (ProvisionMetadata). The meta-data is very important, since it contains (developer defined) quality information like age, reliability, the error of a measurement, etc. The meta-data is organized in a **ContextMetadata** element. It consists of several entries (MetadataEntries attribute), each defined in a name-value element (**ContextMetaProperty**).

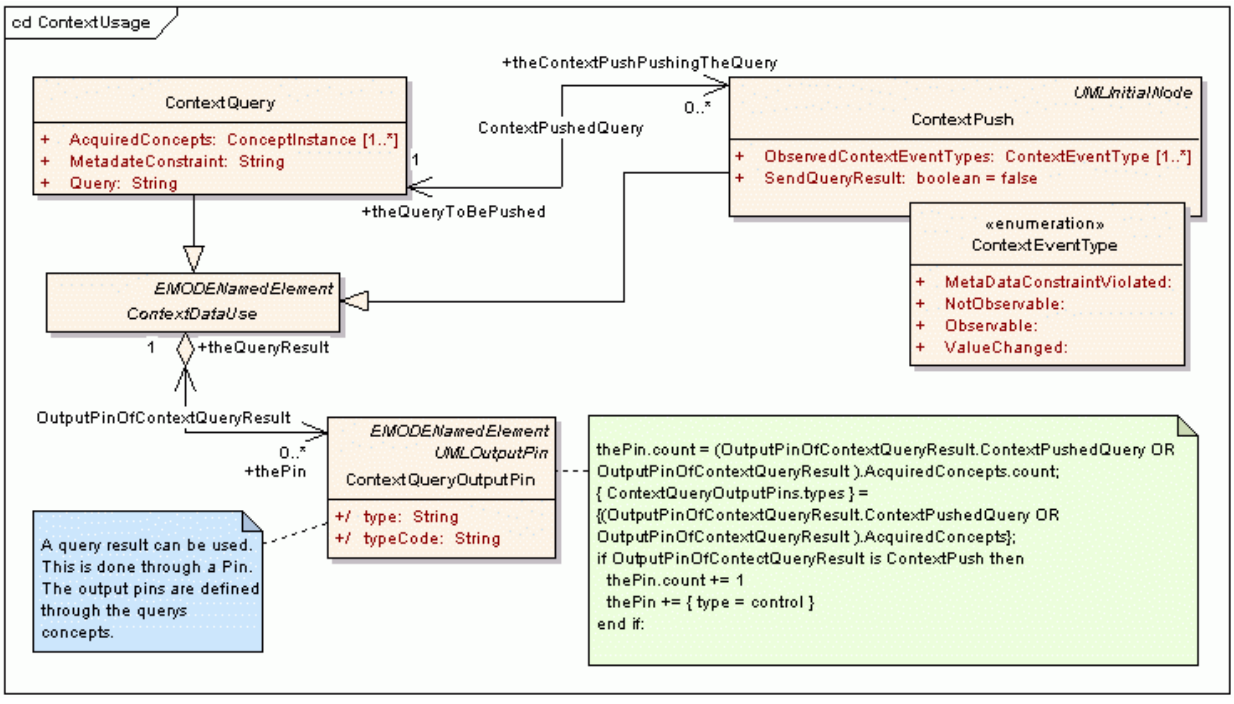


Figure 17: Queries to the context can be used by using them as data input or subscribing to events. The constraint takes care that all the needed pins for a ContextDataUse are created.

A **ContextQuery** has three key attributes. The concepts, which the query returns, are defined with AcquiredConcepts, constraints on the meta data can be formulated with MetadataConstraint (e.g., "a query result must have an error smaller than…"), and the query itself is formulated in the Query attribute. The formulation of the query is hereby done in a language that needs to be specified for the design and runtime, especially in respect to the specification and implementation of the context service. The output of a query is the context data itself - it can be used, e.g., as data input.

A second option for using the context is to subscribe to events rather than the data itself. These events are formulated against the result of a **ContextQuery**. The partitioning of event types is defined in the **ContextEventType** enumeration:

- *MetaDataConstrainViolated*: a quality constraint is not fulfilled anymore,
- *NotObservable*[11]: the context information cannot be provided anymore (e.g., because no providers were found to deliver or the only sensor delivering it is disconnected),
- *Observable*: the context is observable again, and
- *ValueChanged*: the result of the ContextQuery changed.

A **ContextPush** can observe one or more of these events. Via the ContextPushedQuery association, the query is defined that the events should be observed for. It also can be specified (by SendQueryResult) whether just the control flow should be invoked or data should be (re)posted upon event occurrence.

The **ContextDataUse** element is the superclass of **ContextQuery** and **ContextPush**. It resembles their common characteristic that they can be connected to **TaskNodes**. This connection is done via **ContextQueryOutputPins** and **TaskEdges**. For each provided concept and (in case of a **ContextPush**) the control flow, an output pin is generated. These pins can then be connected via **TaskObjectEdges** respectively **TaskControlEdges** (cf. section 2.4.1). The context queries connected hereby meant for delivering input data to a subsequent task are evaluated right before the task starts.

A consequence of this specification is that the context service implementing the query and provision functionality, the application and the context users must all share a common (part of) the DomainConcept model. This reflects the need that everybody involved (providers and users) has to know what the concepts are, the others are talking about.

To further increase flexibility, the context service could be equipped with facilities to dynamically swap providers in and out. Further specification of this has been and will be done in EMODE work package 3 (design and runtime environment).

2.6 EMODECommons

The commons package consists of several supporting elements to improve the quality and effectiveness of modelling. It provides support for library definition and inclusion as well as patters and annotations. Also generic concepts (like the **EMODEElement** – every element in an EMODE model is derived from it) are defined here.

[11] Of course, *NotObservable* and *Observable* could be merged into one element *ObservabilityChanged*, but we think that the two states are very important for developing adaptive applications. It therefore is reasonable to differentiate.

2.6.1 Generic

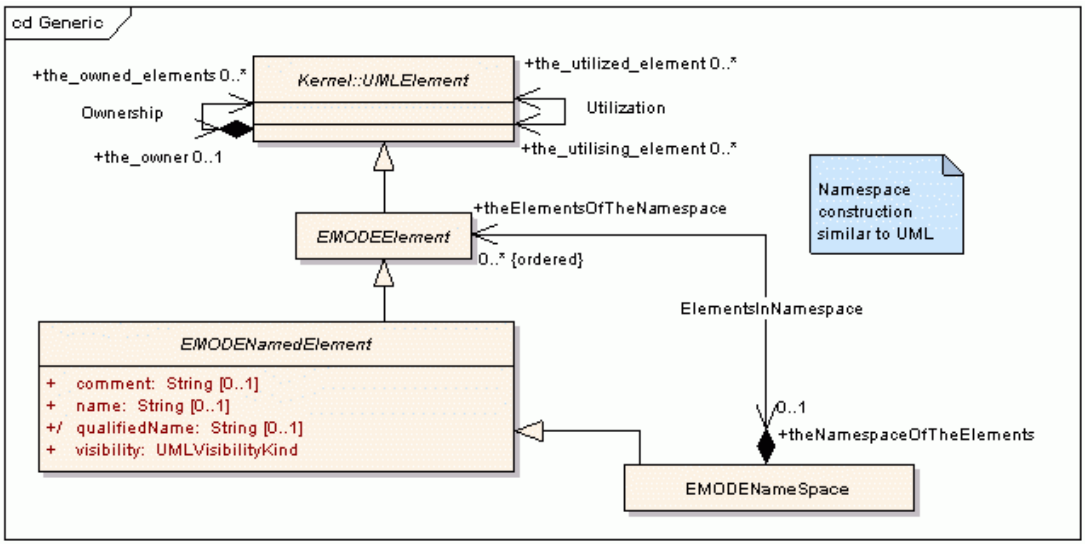


Figure 18: specification of an element and a named element as well as a namespace.

This diagram (Figure 18) depicts the specification of an **EMODEElement**, an **EMODENamedElement** and an **EMODENameSpace**.

Every element in an EMODE model is derived from the **EMODEElement**. For compatibility reasons, this element is generalized to *UML Element* (which will be removed in a future version). An **EMODENamedElement** is a specialization of an **EMODEElement**, and enriched with a name, the possibility to derive a qualifiedName (with namespace prefix) and the visibility of the element (like in UML). It therefore has the same attributes and members as the *UML Named Element*.

An **EMODEElement** may belong to one **EMODENameSpace**, the name space in turn may contain 0 or more elements. It is important to note that it is therefore not possible to reference every element in a namespace by its name, but all elements within a namespace can be enumerated.

2.6.2 Models

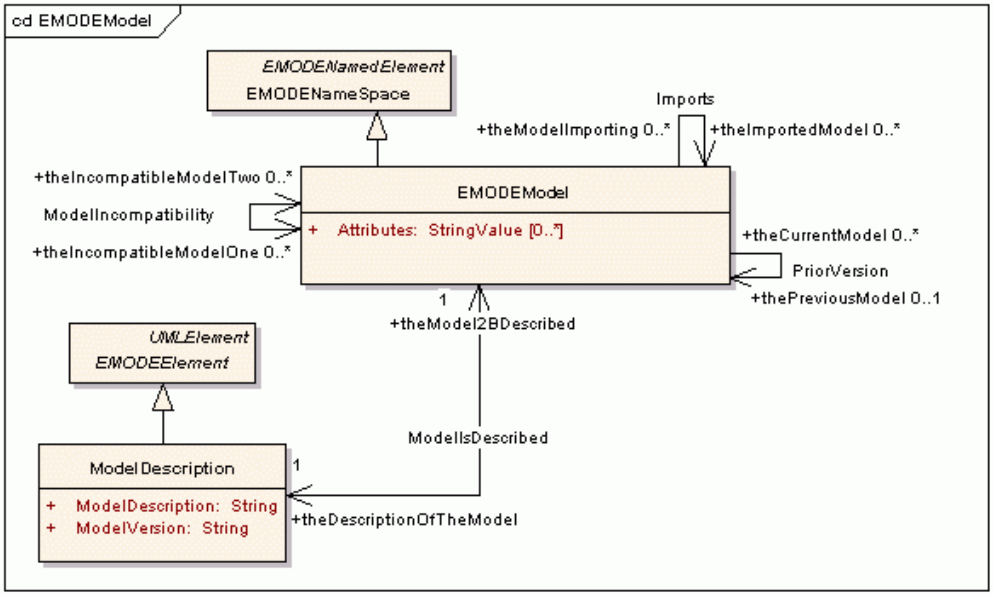


Figure 19: EMODE models can be imported, related for versioning and defined as incompatible, also a description can be given for a model.

An **EMODEModel** is the element representing the model a developer edits. A model essentially is a namespace, i.e. it contains other **EMODEElements**. A model can have attributes and be described via a **ModelDescription** element. Additionally relations between models are possible, as seen in Figure 19. Specifically, we can define:

- the import of a model into another,
- the statement that a model is a version prior to the one referencing it, and
- the incompatibility of models.

2.6.3 Libraries

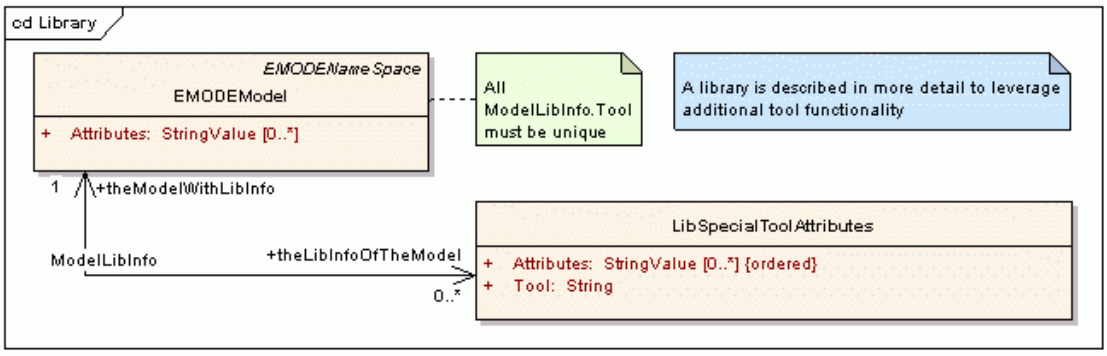


Figure 20: a library is a model with special tool attributes.

Libraries are special models that (as all models can) be imported. The add-on for libraries is that a library designer might choose to, e.g., generate icons or special descriptions for a library. This is facilitated by providing a **LibSpecialToolAttributes** element (depicted in Figure 20). For each tool, such an element can be produced.

2.6.4 Patterns

Design patterns have been successful for quite some time and are used widely in software development. Especially the very comprehensive book by Gamma, Helm, Johnson and Vlissides [Gamma 95] contributed to their popularity. Since it facilitates structured development and reuse of pre-existing solutions, this powerful concept should also be supported when modelling

EMODE applications. We include a pattern diagram in this package. The pattern diagram (Figure 21) depicts the definition of the needed constructs.

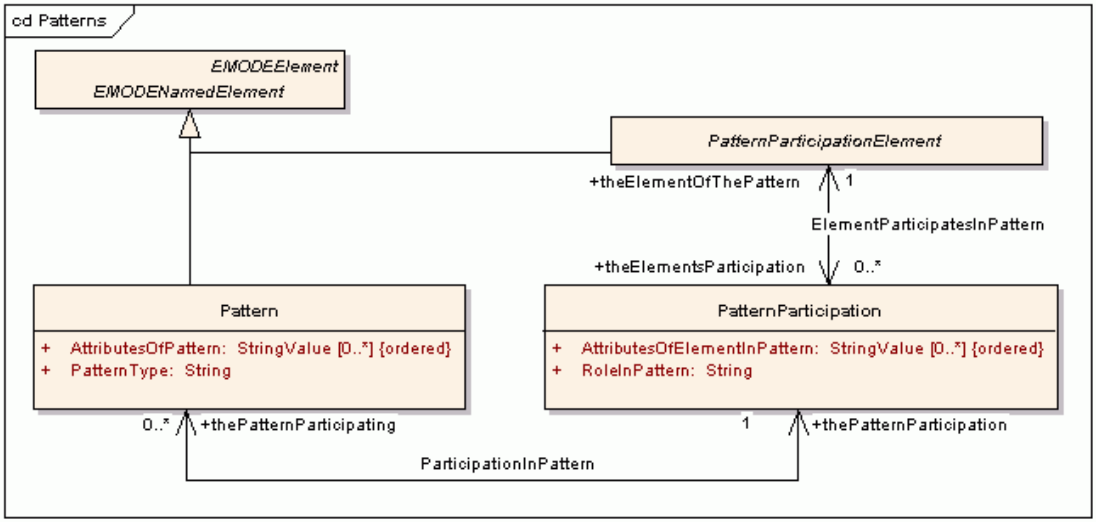


Figure 21: Patterns can be defined across PatternParticipationElements via the PatternParticipation.

The use of patterns can include, but is not limited to, input for transformations, hints for transformations, connection between elements in different packages, documentation purposes …

In Figure 21, the definition of the pattern implementation is shown. An element that participates in a pattern is of type **PatternParticipationElement**. The pattern itself is defined via the **Pattern** element. Finally, the connection between the element to participate and the pattern is made via the **PatternParticipation** element.

Further information like developer defined attributes, a role and type names can be defined by using the AttributesOfPattern attribute of the **Pattern** element.

The elements introduced in the patterns package were designed having transformations in mind. This definition is much more detailed as the one in UML, where there is only one collaboration element.

2.6.5 Annotations

For special tools it is desirable to save information along with the model only relevant for the tool itself. Also for the developer it is useful to comment selected elements of the models. This is facilitated by the **Annotation** element.

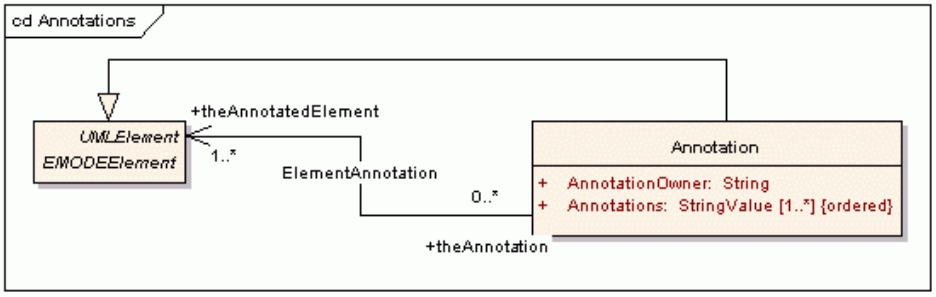


Figure 22: EMODE elements can be annotated.

The Annotation element, as depicted in Figure 22, can hereby be given an owner of the annotation(s) (e.g., the tool name or “developer” for developers, also role names can be envisioned) with the attribute AnnotationOwner. The annotations for that particular owner can be placed in the list Annotations. This list consists of name – value pairs and therefore allows a structuring of the annotations. Contents of annotations could be layout information or additional tool specific information for an element. For general information that is relevant to more than one owner, e.g., graphical layout information, specific formats have to be defined in the process in order to make them interoperable.

The **Annotation** element can be applied to every element within an EMODE model (as defined by the relation to **EMODEElement**). Since an **Annotation** itself is an **EMODEElement**, it can also be annotated.

Transformations might need information added to elements, as well (e.g. markings, as introduced in [MDAGuide03]). This information is not specified for the EMODE project by now, as transformation specification is a separate package. Outcomes of this package, particularly what information is needed by transformations, will be integrated into the meta model.

2.6.6 Storing Graphical Representation Information

Depending on the envisioned design environment, different design information will have to be stored along with the model. This information will be available after the design environment has been specified further in detail (in D3.2). When this information is available, meta model elements will be constructed facilitating its storage.

2.7 Functional Core Adapter

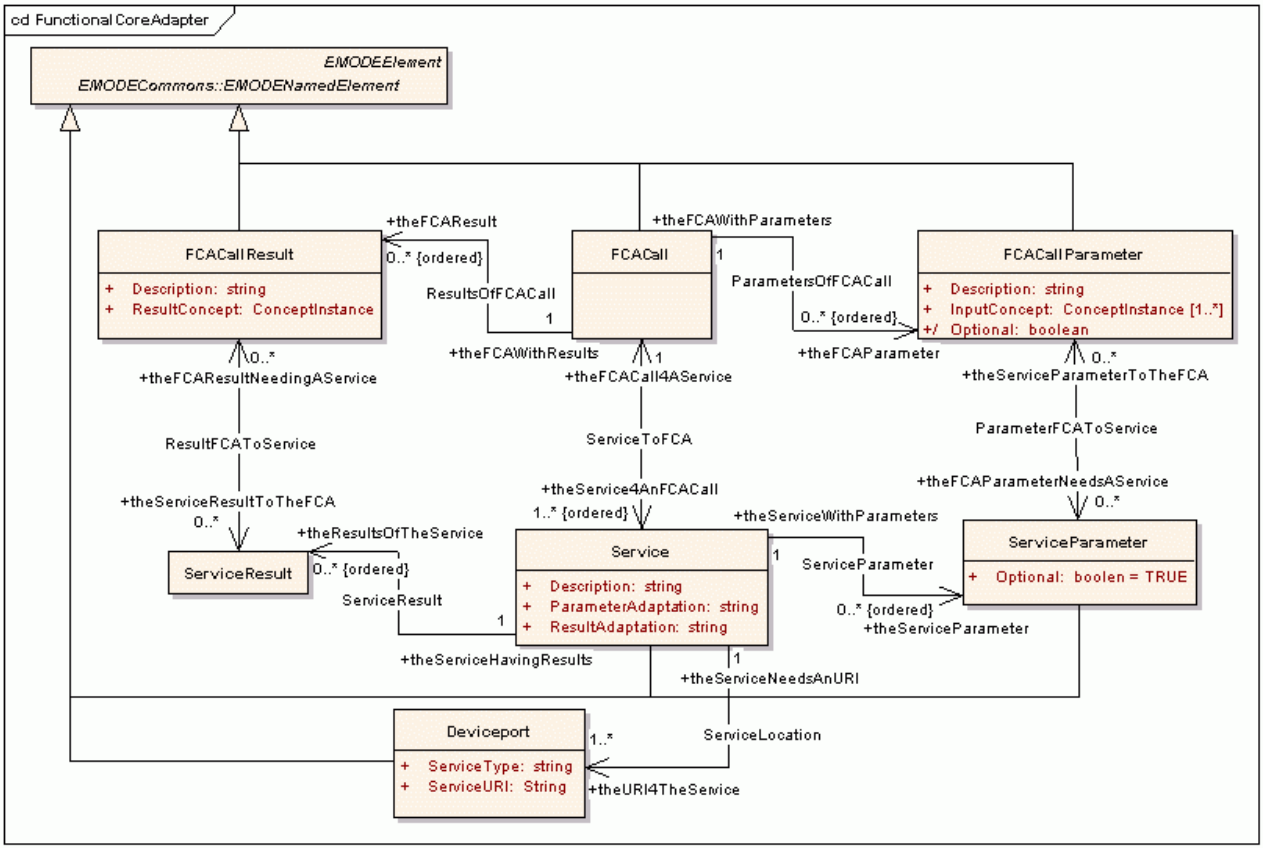


Figure 23: Functional Core Adapter package. It enables mapping of several services to one FCA proxy.

The FunctionalCoreAdapter (FCA) package facilitates a *Service Oriented Architecture* approach. Local and remote services (e.g., calculation logic) can be integrated into an EMODE application by defining it with the elements **Service**, **ServiceParameter** and **ServiceResult**. The term *service* hereby is not to be seen in the strict sense of a web service or alike. It rather implies that functionality is imported from outside the model.

Since a **TaskNode** has input and output concepts, but no support for combined, in- and ouput concepts, this is reflected in the FCA package. Therefore, **Services** and **FCACalls** have separate parameters for in- and output – and not only one result. Besides the parameters, the Service needs one or more location definitions (**Deviceport**), where the EMODE application can access it.

Services integrated through the **Service** element can be accessed through an **FCACall**. This call, as the service, has parameters (**FCACallParameter**) and results (**FCACallResult**). An **FCACall** with its parameters and results can be mapped to several **Services**. Therefore, each component is connected independently (**Service** or **FCACall**, their parameters and results). The developer can hereby define the mapping of service parameters and results to FCA parameters and results in detail. Through this mapping, also the Optional attribute of the **FCACallParameter** is derived.

With this package, an abstraction from the underlying logic in terms of implementation is accomplished. It does not matter whether an **FCACall** is implemented “within” the EMODE application and therefore accessed locally or whether it is located on a distant server. Through the redundant definition of several services for one **FCACall** by the developer, network problems and the like do not have a big impact on the EMODE application.

A possible extension of this could be to enable dynamic mapping of **Services** to **FCACalls**. Possibilities for realization include ontologies or predefined service classes. But this is out of the scope of the EMODE project.

2.8 Modality

For formulating constraints against modalities, the developer needs to define profiles. These profiles, the provisions of the current platform are checked against. Through this the requirements formulated by the applications is matched against what is available through the current platform. Such a profile is called a Modality Restriction Profile.

Similar to CC/PP (Composite Capabilities/Preference Profiles) [CCPP], a common set of attributes (a vocabulary) is defined. A (client) platform (e.g., in the case of CC/PP a cellular phone) uses this set of attributes to formulate its capabilities and user preferences (e.g., the screen size, sound output, etc.) in a profile The partner receiving such a profile can then adapt its output to suite the profile best.

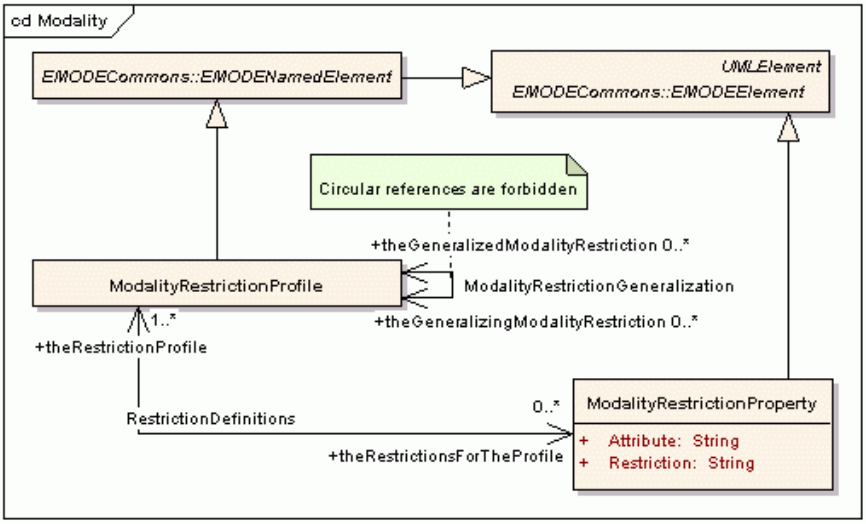


Figure 24: Modality package defining elements for specification of restriction profiles.

A similar idea, but more lightweight, was followed here. Runtime and application can formulate their capabilities and needs for interaction by defining profiles they support. Such a profile is specified, as depicted in Figure 24, with the **ModalityRestricionProfile** element. It consists of 0 or more **ModalityRestrictionProperties**. A **ModalityRestrictionProperty** in turn defines a restriction on a single verb of the common grammar (e.g., the verb “screen size x”). This grammar will be developed during the project and an EMODE grammar will be part of D2.4.

After application and runtime formulated their supported profiles, these profiles have to be compared. Therefore, the grammar must be shared among them. The result of the comparison, i.e. if the needed **ModalityRestrictionProfile** is supported, can then be used in, e.g., guard conditions or as input to decisions.

So in contrast to CC/PP, where the client only formulates CC/PP profiles, the profile is here used twice. The developer specifies requirements on the platform and modalities. The platform describes its capabilities with the profile. This facilitates a more open adaptation process, because both parties formulate their needs.

3 Summary

3.1 Review of Design Corner Stones

In section 1.1, the corner stones for design of the meta model were introduced. These are reflected in the current version of the meta model. In the following a short description of where the corner stones can be seen is given:

Full control over UI look and feel: Through the possibility to define any AUI produced and/or used for the AUI refinement process (cf. the tree of AUI refinement in Figure 9), the developer has full control of the final layout of the application. Particularly the developer is placed in the position to explicitly model the UI shown on a specific modality.

MDA support: The meta model developed here is MOF 1.4 compliant and can be used with model transformations, as will be defined in D2.3. Further model elements needed to fully support EMODE transformations will be identified within D2.3 and D3.3 and integrated into this meta model.

Transformations especially apply to the DialogueSpace model as it heavily relies on them to reduce development effort. The models produced and crossed are hinted by the modality model. This refines the MDA approach for EMODE by providing a fine grained view on the platform independence.

In [Koch02] it is argued that as much information as possible should be kept at a level as abstract possible. This is facilitated through the structure of the DialogueSpace package: keeping as much information as possible in an DialogueSpace model as modality independent as possible. This approach can further be supported by providing the developer with facilities to easily define how information from an abstract level can be concretized; e.g., by easy definition of transformations.

Improved integration of UI and system development: The task model serves, as explained in section 1.1, as a basis for improving the integration of UI and system development.

Support for Service Oriented Architecture: Application logic is handled as services and referenced through the FCA model. These internal and other external services are treated the same in the context of the FCA model. Through this and the architecture of the runtime environment (which is part of D3.1), a service oriented approach is facilitated and functionality can be composed of separate modules.

Context-awareness: This can be reached on different paths: 1) taking context into account when modelling, 2) defining adaptation rules or 3) defining constraints for automatic adaptation[12]. The first path is supported by enabling the developer to query the context and use it as input data for computations or decisions. Nevertheless, it is somewhat restricted, because every possible adaptation has to be formulated out fully by the developer. The second path is even more restricted, since the context cannot easily be used as input data and therefore was not a design aim of the meta model. And the third path is supported through the usage of modelling. But to fully enable it, runtime transformations and verification rules must be integrated. The degree of adaptivity can even be improved further by making the meta model more open (e.g., through elaboration of the DomainConcept package) and relying on more automatic adaptation processes.

Semantically close to UML: Finally, the meta model was designed by starting with derivation of elements from UML elements (which were defined within EUML [EUML]). Semantic proximity to UML is conserved, e.g., through TaskNodes being derived from UML Activity Nodes. In order to improve readability and implementability the number of elements in the meta model is kept much smaller than in UML. Also, not all elements are derived from UML elements, since UML elements often carry members and attributes that are superfluous in the context of EMODE.

3.2 Conclusion and Outlook

Modelling of multimodal, context aware, adaptive applications is supported by the meta model presented in this document. Modular development and model driven architecture are supported in particular through using models and relying on transformations, which will be the focus of D2.3.

Throughout the EMODE project, this meta model will be revised and new insights integrated – the final version to be presented in D2.4. Semantic relation to UML will be conserved (e.g., in the TaskNodes, which are like UML Activity Nodes), but UML constructs will be removed in order to remove elements not needed and improve readability.

[12] This does not imply how exactly the adaptation is implemented (runtime or design time), but rather how the developer “feels” when modelling it.

References

[Braun 03] Braun, E. & Hartl, A. (2003),Device-Spanning Multimodal User Interfaces, *in* 'UbiComp 2003, The Fifth International Conference on Ubiquitous Computing, Adjunct Proceedings', pp. 147--148.

[Brockmans 04] Brockmans, S.; Volz, R.; Eberhart, A. & Löffler, P. (2004),Visual Modeling of OWL DL Ontologies Using UML., *in* 'International Semantic Web Conference', pp. 198-213.

[Calvary 03] Calvary, G.; Coutaz, J.; Thevenin, D.; Limbourg, Q.; Bouillon, L. & Vanderdonckt, J. (2003), 'A Unifying Reference Framework for Multi-Target User Interfaces', *Interacting with Computers* **15**(3), 289-308.

[Calvary 04] Calvary, G.; Coutaz, J.; Dâassi, O.; Balme, L. & Demeure, A. (2004),Towards a New Generation of Widgets for Supporting Software Plasticity: The "Comet", *in* Rémi Bastide; Philippe Palanque & Jörg Roth, ed.,'Engineering Human Computer Interaction and Interactive Systems: Joint Working Conferences EHCI-DSVIS 2004', Springer-Verlag GmbH, , pp. 306--326.

[CAMELEON] http://giove.cnuce.cnr.it/cameleon.html

[CCPP] http://www.w3.org/TR/CCPP-struct-vocab/

[Djuric 05] Djuric, D.; Gasevic, D. & Devedzic, V. (2005), 'Ontology Modeling and MDA.', *Journal of Object Technology* **4**(1), 109-128.

[EMODE] http://www.emode-projekt.de

[EMODE 05] Consortium, E. (2005),'Projektantrag zur Forschungsoffensive "Software Engineering 2006"'.

[EUML] IKV++ Technologies AG,'medini component modeler'.

[Gamma 95] Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J.Kernighan, B.W., ed. (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

[Hartl 03] Hartl, A. (2003),A Widget-Based Approach for Creating Voice Applications, *in* 'Proceedings of MobileHCI - Physical Interaction (PIO3) - Workshop on Real World User Interfaces', pp. 7-10.

[IBM 05] IBM & Sandpiper Software (2005),'Ontology Definition Metamodel, Third Revised Submission to OMG'.

[Jeckle] http://www.jeckle.de/umllinks.htm#tutorials

[Kath 03] Kath, O.; Holz, E. & Born, M. (2003), *Softwareentwicklung mit UML 2*, Addison-Wesley.

[Koch 02] Koch, T.; Uhl, A. & Weise, D. (2002),'Model Driven Architecture'.

[MDAGuide 03] Miller, J. & Mukerji, J. (2003),'MDA Guide Version 1.0.1', OMG.

[Mori 03] Mori, G.; Paternó, F. & Santoro, C. (2003),Tool support for designing nomadic applications, *in* 'IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces', ACM Press, New York, NY, USA, pp. 141--148.

[Myers 00] Myers, B.; Hudson, S.E. & Pausch, R. (2000), 'Past, Present, and Future of User Interface Software Tools', *ACM Transactions on Computer-Human Interaction (TOCHI)* **7**(1), 3--28.

[OCL2] http://www.omg.org/technology/documents/formal/ocl.htm

[ODMRFP] http://www.omg.org/cgi-bin/doc?ad/2003-3-40

[OWL] http://www.w3.org/TR/owl-semantics/

[Paterno 01] Paterno, F. (2001), *Task Models in Interactive Software Systems*, World Scientific Publishing Co..

[Pilone 05] Pilone, D.Pilone, D., ed. (2005), *UML 2.0 In A Nutshell*, O'Reilly.

[Thevenin 99] Thevenin, D. & Coutaz, J. (1999),Plasticity of User Interfaces: Framework and Research Agenda, *in* A.M. Sasse & C. Johnson, ed.,'Proceedings of the IFIP Conference on Human-Computer Interaction (INTERACT99)', IOS Press, , pp. 110--117.

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

[Usixml 04] Limbourg, Q.; Vanderdonckt, J.; Michotte, B.; Bouillon, L.; Florins, M. & Trevisan, D. (2004),UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces, *in* K. Luyten; M. Abrams; Q. Limbourg & J. Vanderdonckt, ed.,'Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" (Gallipoli, May 25, 2004)', pp. 55-62.

[W3C 04] http://www.w3.org/TR/2004/REC-owl-guide-20040210/#Datatypes1

Appendix A: Automatically Generated Meta Model Documentation

Model Detail

EMODESpecific

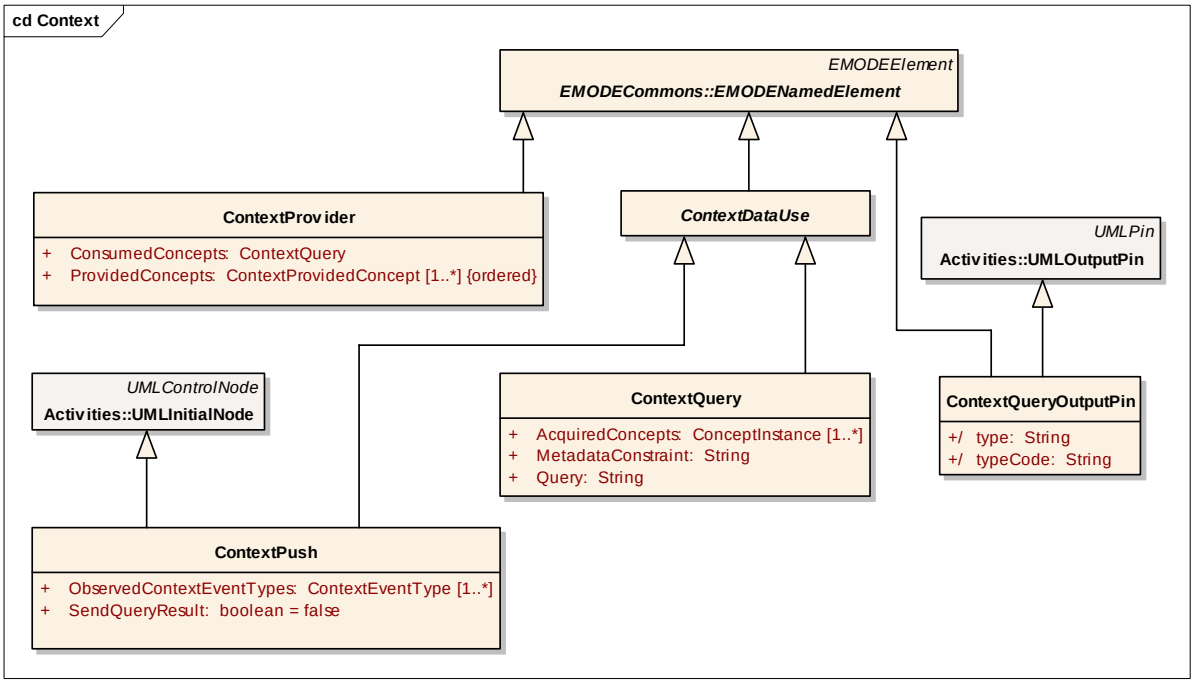
Type: **Package**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODE
Detail: *Created on* 08.03.2006 14:10:19. *Last modified on* 19.04.2006 14:55:49
GUID: {4CAC032A-8020-4778-B45E-DDCCB3EA7738}

EMODESpecific - *(Class diagram)*
Created By: J. Hößler on 08.03.2006
Last Modified: 21.04.2006
Version: 1.0. False
GUID: {44CD051D-655E-47cd-B1FD-289EBDA4E49B}

Context

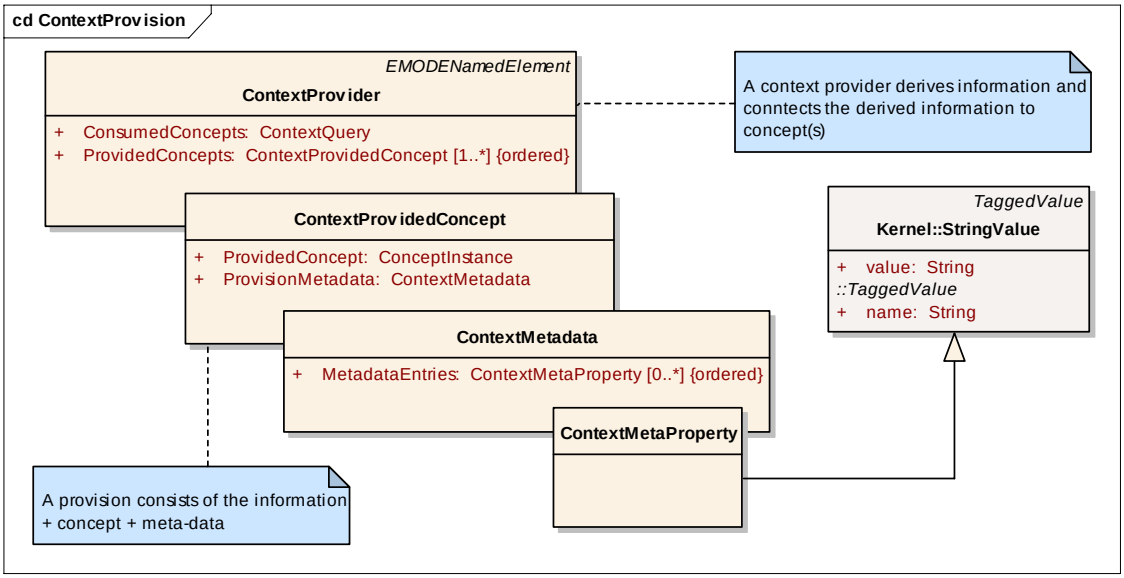
Type: **Package**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODESpecific
Detail: *Created on* 07.03.2006 17:07:48. *Last modified on* 08.03.2006 14:11:21
GUID: {CF810FDF-457C-4547-8FC7-4B8BD29C55E2}

Context - *(Class diagram)*
Created By: J. Hößler on 07.03.2006
Last Modified: 21.06.2006
Version: 1.0. False
GUID: {56C87588-7B0D-40bc-9C62-C4ADDEE63E85}



ContextProvision - *(Class diagram)*

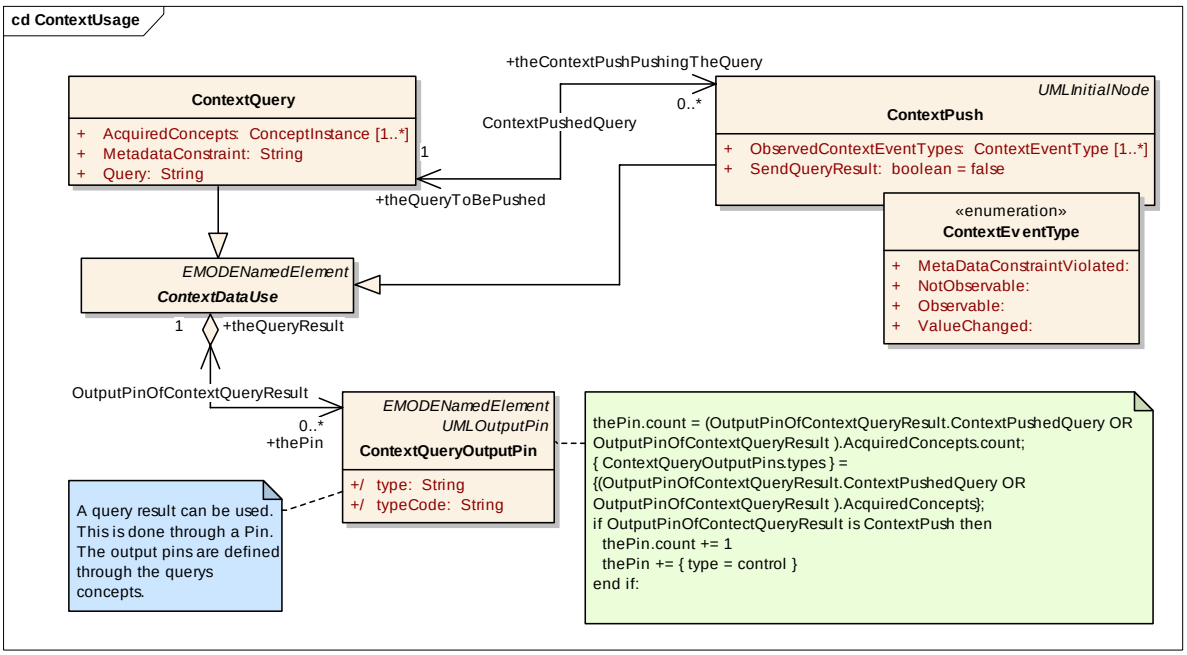
Created By: Alexander Behring on 30.05.2006
Last Modified: 16.06.2006
Version: 1.0. False
GUID: {9820129D-8EF8-45fd-A2D7-AAF928D9EC24}



ContextUsage - *(Class diagram)*

Created By: Alexander Behring on 18.04.2006
Last Modified: 21.06.2006
Version: 1.0. False

GUID: {65737C98-7F55-4602-95ED-5E7B785F9B69}



ConceptProviderMetaData

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 12.04.2006. *Last modified on*.12.04.2006.
GUID: {13AB2E4E-9E46-462c-B966-EBA5A639DEEC}

Metadata associated with a provided concept instance. Features things like when acquired, etc.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Attributes

Attribute	Notes	Constraints and tags

Attribute	Notes	Constraints and tags
Reliability float Public		*Default:*

ContextDataUse

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 20.06.2006. *Last modified on*.20.06.2006.
GUID: {E9C23D7B-536A-48bf-9297-239AB9220FD8}

The data of a context data can be used in different ways

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Aggregation OutputPinOfContextQueryResult Bi-Directional	Public thePin ContextQueryOutputPin	Public theQueryResult ContextDataUse	
Generalization Source -> Destination	Public ContextPush	Public ContextDataUse	
Generalization Source -> Destination	Public ContextQuery	Public ContextDataUse	
Generalization Source -> Destination	Public ContextDataUse	Public EMODENamedElement	

ContextEventType

Type: **Enumeration**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 18.04.2006. *Last modified on*.18.04.2006.
GUID: {B003B8A0-674B-4a3e-BC93-CAFE1EF88B20}

The types of ContextEvents that can be thrown

Custom Properties

- isActive =

Attributes

Attribute	Notes	Constraints and tags
MetaDataConstraint Violated Public		*Default:*
NotObservable Public		*Default:*
Observable Public		*Default:*
ValueChanged Public		*Default:*

ContextEventType

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 18.04.2006. *Last modified on*.18.04.2006.
GUID: {44FA4AE2-1D4F-405d-A008-72B1D1B0EAC2}

Types of events that can be thrown by a ContextProvider

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

ContextMetadata

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 27.04.2006. *Last modified on*.21.07.2006.
GUID: {1CC3D91A-D448-4992-A07D-723281B112FF}

A bag of metainformation for context information

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Attributes

Attribute	Notes	Constraints and tags
MetadataEntries ContextMetaProperty Public [0..*]	The name-value pairs of the metadata	*Default:*

ContextMetaProperty

Type: **Class StringValue**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 27.04.2006. *Last modified on*.27.04.2006.
GUID: {7F7B3E39-4331-40ff-937B-8E06303CE37A}

A name value pair containing a context information meta data

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ContextMetaProperty	Public StringValue	

ContextProvidedConcept

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 27.04.2006. *Last modified on*.27.04.2006.
GUID: {9FC19646-2D2C-4d12-AF29-D546882C82A4}

A description of a provision of a concept by an entity of the context

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
NoteLink Source -> Destination	Public Note	Public ContextProvidedConcept	

Attributes

Attribute	Notes	Constraints and tags
ProvidedConcept ConceptInstance Public	The concept which is provided	*Default:*
ProvisionMetadata ContextMetadata Public	Matedata associated with the provided concept	*Default:*

ContextProvider

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 07.03.2006. *Last modified on*.30.05.2006.
GUID: {0A515E07-F78C-477a-BE80-05282CA2B27B}

A ContextProvider is providing information. The information is of a certein type/concept. The view is that a provider encapsulates an engine - i.e. it can have several input and output channels. Since it collects its needed information itself, the input channels are not modelled.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes

Connector	Source	Target	Notes
Association Source -> Destination	Public ContextProvider	Public ContextProvider	
Generalization Source -> Destination	Public ContextProvider	Public EMODENamedElement	
NoteLink Source -> Destination	Public Note	Public ContextProvider	

Attributes

Attribute	Notes	Constraints and tags
ConsumedConcepts ContextQuery Public	Concepts consumed by the provider	*Default:*
ProvidedConcepts ContextProvidedConcept Public [1..*]	The concepts, this provider provides along with provision infromation	*Default:*

ContextPush

Type: **Class ContextDataUse, UMLInitialNode**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 15.05.2006. *Last modified on*.15.05.2006.
GUID: {156874D9-13F5-4598-B826-823043F2D3D9}

The context server pushes the information through this channel, that an event of the specified type is raised.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ContextPush	Public ContextDataUse	
Association ContextPushedQuery Bi-Directional	Public theContextPushPushingTheQuery ContextPush	Public theQueryToBePushed ContextQuery	Connects the push definition with the context query for the push
Generalization Source -> Destination	Public ContextPush	Public UMLInitialNode	

Attributes

Attribute	Notes	Constraints and tags
ObservedContextEventTypes ContextEventType Public [1..*]	The context event types on which the system should react	*Default:*
SendQueryResult boolean Public false	Send the (new) query result through this channel as well	*Default:* false

ContextQuery

Type: **Class ContextDataUse**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 21.04.2006. *Last modified on*.20.06.2006.
GUID: {D55FE4A5-7E90-4613-85C8-34EFD9E86B42}

A query defines what information is retreived - it should not define how, since this is modeled with providers. It is applied to the concept/context model of the application.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ContextQuery	Public ContextDataUse	
Association ContextPushedQuery Bi-Directional	Public theContextPushPushingTheQuery ContextPush	Public theQueryToBePushed ContextQuery	Connects the push definition with the context query for the push

Attributes

Attribute	Notes	Constraints and tags
AcquiredConcepts ConceptInstance Public [1..*]	The concepts that this query asks for and returns to the "caller"	*Default:*
MetadataConstraint String Public	The constraints on the metadata - can include values, ranges with the operators <,> and =	*Default:*
Query String Public	This query might be defined in a language like rdql, it should also include constraints on the metadata of the elements. In order to be functional on concepts without context, too, non existing metadata values are evaluated as a NULL entry.	*Default:* [isStatic = false]

ContextQueryOutputPin

Type: **Class EMODENamedElement, UMLOutputPin**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Context *Keywords:*
Detail: *Created on* 12.05.2006. *Last modified on*.12.05.2006.

GUID: {B669E3A0-AF29-449d-8D31-8AB8DECE2AF8}

Implements a pin as an output of a context query

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
NoteLink Source -> Destination	Public Note	Public ContextQueryOutputPin	
Aggregation OutputPinOfContextQueryResult Bi-Directional	Public thePin ContextQueryOutputPin	Public theQueryResult ContextDataUse	
Generalization Source -> Destination	Public ContextQueryOutputPin	Public UMLOutputPin	
NoteLink Source -> Destination	Public Note	Public ContextQueryOutputPin	
Generalization Source -> Destination	Public ContextQueryOutputPin	Public EMODENamedElement	

Attributes

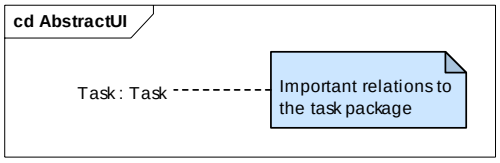
Attribute	Notes	Constraints and tags
type String Public	Overwrites	*Default:*

Attribute	Notes	Constraints and tags
typeCode String Public	Overwrites	*Default:*

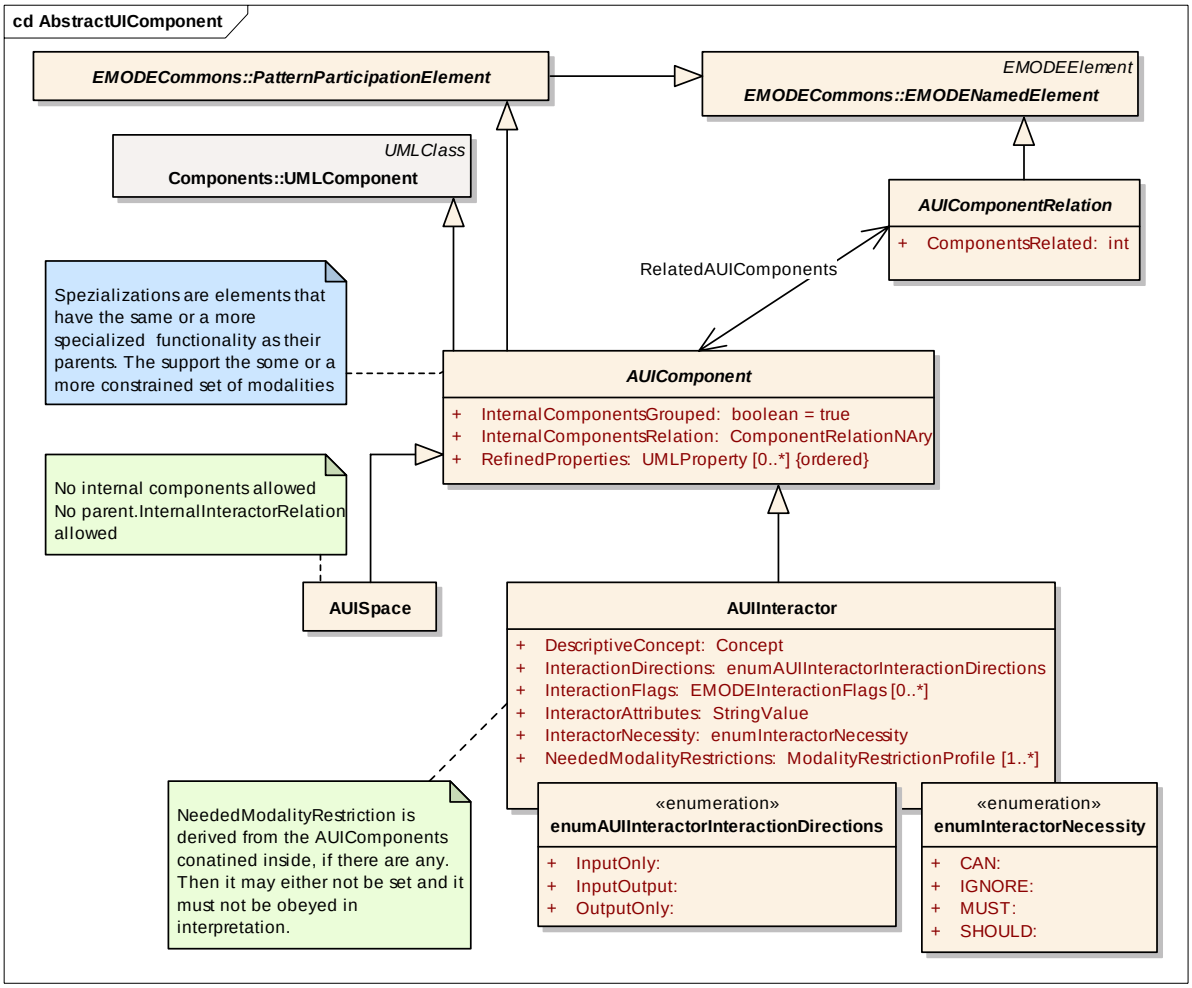
DialogueSpace

Type: **Package**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODESpecific
Detail: *Created on* 07.03.2006 17:18:16. *Last modified on* 15.03.2006 11:24:05
GUID: {1C8B0298-4C01-4825-803B-1063AE3DCB71}

AbstractUI - *(Class diagram)*
Created By: J. Hößler on 07.03.2006
Last Modified: 03.06.2006
Version: 1.0. False
GUID: {7C069CF3-DE2A-498a-A2AD-6A1316155F9C}

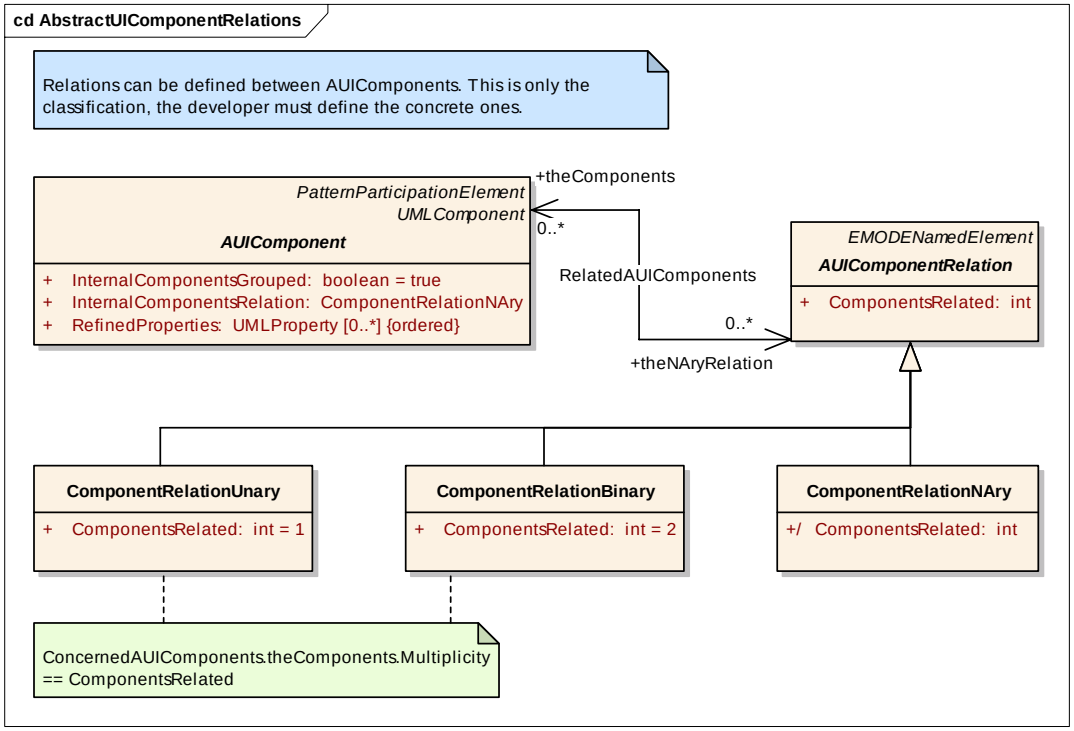


AbstractUIComponent - *(Class diagram)*
Created By: Andreas Petter on 22.03.2006
Last Modified: 04.07.2006
Version: 1.0. False
GUID: {F5823867-F28C-46bd-ACF3-D32F40919B05}
incomplete, subject to change



AbstractUIComponentRelations - *(Class diagram)*

Created By: Alexander Behring on 14.06.2006
Last Modified: 07.07.2006
Version: 1.0. False
GUID: {7B2E120C-D5B5-4dfa-8A76-752B01199EE1}



AUIComponent

Type: **Class PatternParticipationElement, UMLComponent**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DialogueSpace *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.05.07.2006.
GUID: {D65FC2CC-DDA9-40de-A6F7-963045E3916B}

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public AUIInteractor	Public AUIComponent	
Generalization Source -> Destination	Public AUIComponent	Public UMLComponent	

Connector	Source	Target	Notes
Generalization Source -> Destination	Public AUIComponent	Public PatternParticipation Element	
Association RelatedAUIComponents Bi-Directional	Public theNAryRelation AUIComponentRelation	Public theComponents AUIComponent	The component involved in a relation
NoteLink Source -> Destination	Public Note	Public AUIComponent	
Generalization Source -> Destination	Public AUISpace	Public AUIComponent	

Attributes

Attribute	Notes	Constraints and tags
InternalComponents Grouped boolean Public true	Whether this container should realize a grouping functionality on the components.	*Default:* true
InternalComponents Relation ComponentRelationNAry Public	Describes the relation the AUIComponents which are contained in this component have. An empty element dontes none.	*Default:*
RefinedProperties UMLProperty Public [0..*]	The properties, which are refined in the process of transformations	*Default:*

AUIComponentRelation

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DialogueSpace *Keywords:*
Detail: *Created on* 27.04.2006. *Last modified on*.05.07.2006.

GUID: {BDDECA98-D14E-4fb2-BD44-929FC92FEA58}

A relation between one or more AUI components

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ComponentRelation NAry	Public AUIComponentRel ation	
Generalization Source -> Destination	Public ComponentRelation Unary	Public AUIComponentRel ation	
Generalization Source -> Destination	Public AUIComponentRel ation	Public EMODENamedEle ment	
Generalization Source -> Destination	Public ComponentRelation Binary	Public AUIComponentRel ation	
Association RelatedAUICompone nts Bi-Directional	Public theNAryRelation AUIComponentRel ation	Public theComponents AUIComponent	The component involved in a relation

Attributes

Attribute	Notes	Constraints and tags
ComponentsRelated int Public	The number of related interactors	*Default:*

AUIInteractor

Type: **Class AUIComponent**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DialogueSpace *Keywords:*
Detail: *Created on* 22.03.2006. *Last modified on*.05.07.2006.
GUID: {DACF0A51-5229-460f-862E-E79E8FEAACD3}

Interactor is the superclass of all interactors. Basicly there exist 3 types of interactors: Interactors, that are used for input, interactors that are used for output and containers, which organize interactors.
An interactor itself can be an abstraction being composed of multiple interactors itself. I.e. it represents a more complex interactor being able to manipulate more complex structures.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public AUIInteractor	Public AUIComponent	
NoteLink Source -> Destination	Public Note	Public AUIInteractor	
Association TaskIsEnactedBy Bi-Directional	Public theTaskNode2BeEnacted TaskNode	Public theAUIInteractor2EnactTheTask AUIInteractor	Describes the connection between task nodes and AUI components, which the interaction layout. This is a tight relationship, since the one (tasks) describe timely behaviour, whereas the other describe the layout.

Attributes

Attribute	Notes	Constraints and tags

Attribute	Notes	Constraints and tags
DescriptiveConcept Concept Public	The concept, which describes the interactor best. If no concept is given, the developer will need to take care of the description herself.	*Default:* [isStatic = false]
InteractionDirections enumAUIInteractorInteractionDirections Public	The way (directions) of interaction with the user	*Default:*
InteractionFlags EMODEInteractionFlags Public [0..*]	Interaction properties given to this component	*Default:*
InteractorAttributes StringValue Public	Properties the interactor can have	*Default:*
InteractorNecessity enumInteractorNecessity Public	(see enumInteractorNecessity)	*Default:* [isStatic = false]
NeededModalityRestrictions ModalityRestrictionProfile Public [1..*]	The modality restrictons that need to be enforced for this component to be reificatable	*Default:* [isStatic = false]

AUIInteractorInteractionWays

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DialogueSpace *Keywords:*

Detail: *Created on* 27.04.2006. *Last modified on*.05.07.2006.
GUID: {31DAD375-3AFA-4496-8FF0-C9BDC4625FF9}

Describes what ways of interaction the interactor uses with the user - not taking into accout the incitataion automatically generated by the system.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

AUISpace

Type: **Class AUIComponent**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DialogueSpace *Keywords:*
Detail: *Created on* 02.06.2006. *Last modified on*.05.07.2006.
GUID: {C6C52A15-2BD9-4385-9CB4-394691E33ABE}

This component serves as a placeholder, where other interactors can be placed in.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public AUISpace	Public AUIComponent	
NoteLink Source -> Destination	Public Note	Public AUISpace	
Association TaskIsInteractedAt Bi-Directional	Public theTaskToBeInteracted TaskNode	Public theSpace4Interaction AUISpace	Details, where the interaction should take place

ComponentRelationBinary

Type: **Class AUIComponentRelation**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DialogueSpace *Keywords:*
Detail: *Created on* 27.04.2006. *Last modified on*.05.07.2006.
GUID: {4BA21F31-A692-44fe-837A-1D39353959BB}

A directed relation between two components

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ComponentRelation Binary	Public AUIComponentRel ation	
NoteLink Source -> Destination	Public Note	Public ComponentRelation Binary	

Attributes

Attribute	Notes	Constraints and tags
ComponentsRelated int Public Const 2		*Default:* 2

ComponentRelationNAry

Type: **Class AUIComponentRelation**
Status: Proposed. Version 1.0. Phase 1.0.

Package: DialogueSpace *Keywords:*
Detail: *Created on* 27.04.2006. *Last modified on*.05.07.2006.
GUID: {DCA48D11-A7E7-45f2-B3F2-BEE55635508D}

A n-ary relation between interactors

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ComponentRelation NAry	Public AUIComponentRel ation	

Attributes

Attribute	Notes	Constraints and tags
ComponentsRelated int Public	Is the currently active numer of related components	*Default:*

ComponentRelationUnary

Type: **Class AUIComponentRelation**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DialogueSpace *Keywords:*
Detail: *Created on* 27.04.2006. *Last modified on*.05.07.2006.
GUID: {9146DE75-C3E8-4c9d-820F-803F8B35D063}

An attribute/mark attached to an interactor. The attached mark is a relation.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ComponentRelation Unary	Public AUIComponentRel ation	
NoteLink Source -> Destination	Public Note	Public ComponentRelation Unary	

Attributes

Attribute	Notes	Constraints and tags
ComponentsRelated int Public Const 1		*Default:* 1

enumAUIInteractorInteractionDirections

Type: **Enumeration**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DialogueSpace *Keywords:*
Detail: *Created on* 27.04.2006. *Last modified on*.05.07.2006.
GUID: {0A40651F-9327-43e6-A276-EB2B86AE07EE}

Describes what directions of interaction the interactor uses with the user - not taking into accout the incitataion automatically generated by the system.

Custom Properties

- isActive =

Attributes

Attribute	Notes	Constraints and tags

Attribute	Notes	Constraints and tags
InputOnly Public		*Default:*
InputOutput Public		*Default:*
OutputOnly Public		*Default:*

enumInteractorNecessity

Type: **Enumeration**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DialogueSpace *Keywords:*
Detail: *Created on* 03.04.2006. *Last modified on*.05.07.2006.
GUID: {97D5F87C-5F95-449b-A0E3-E3C559F80BD3}

How necessary an interactor is in order to complete the task and have a good UI

Custom Properties

- isActive =

Attributes

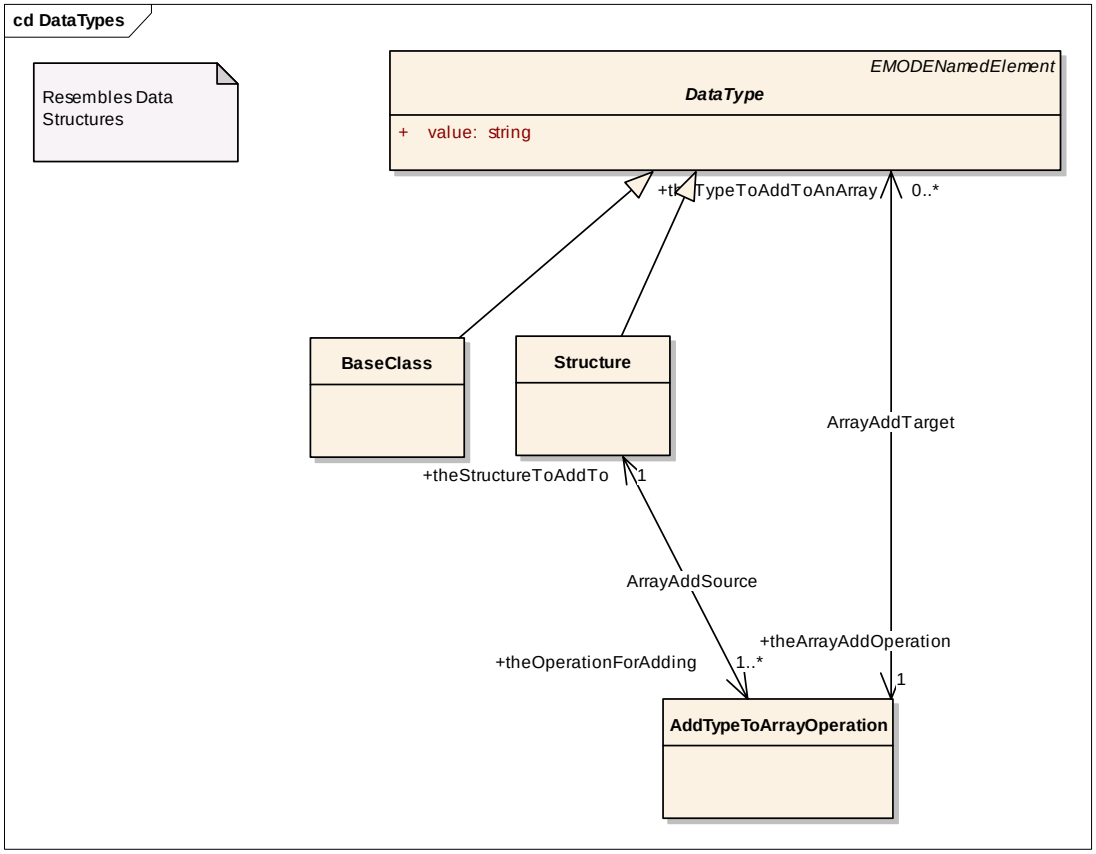
Attribute	Notes	Constraints and tags
CAN Public		*Default:*

Attribute	Notes	Constraints and tags
IGNORE Public		*Default:*
MUST Public		*Default:*
SHOULD Public		*Default:*

DomainConcept

Type: **Package**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODESpecific
Detail: *Created on* 24.03.2006 11:09:28. *Last modified on* 24.03.2006 11:10:59
GUID: {8BA83FDD-966C-48e1-AED2-DAA6F360274D}

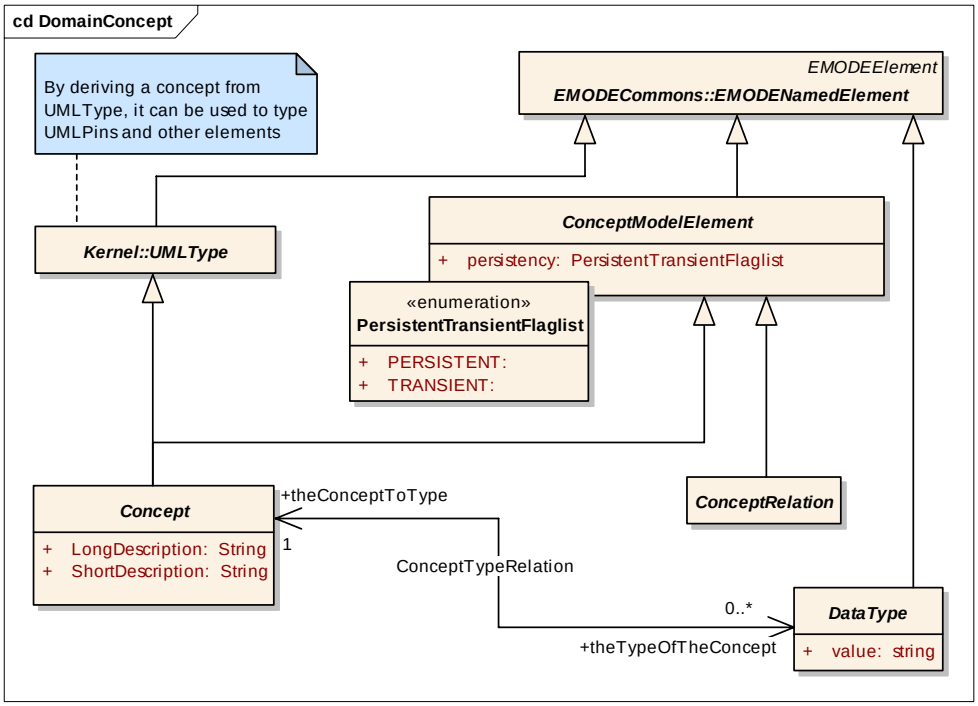
DataTypes - *(Class diagram)*
Created By: Alexander Behring on 29.03.2006
Last Modified: 29.03.2006
Version: 1.0. False
GUID: {49636929-1B2E-49ca-8C58-644130B7DD1D}



DomainConcept - *(Class diagram)*

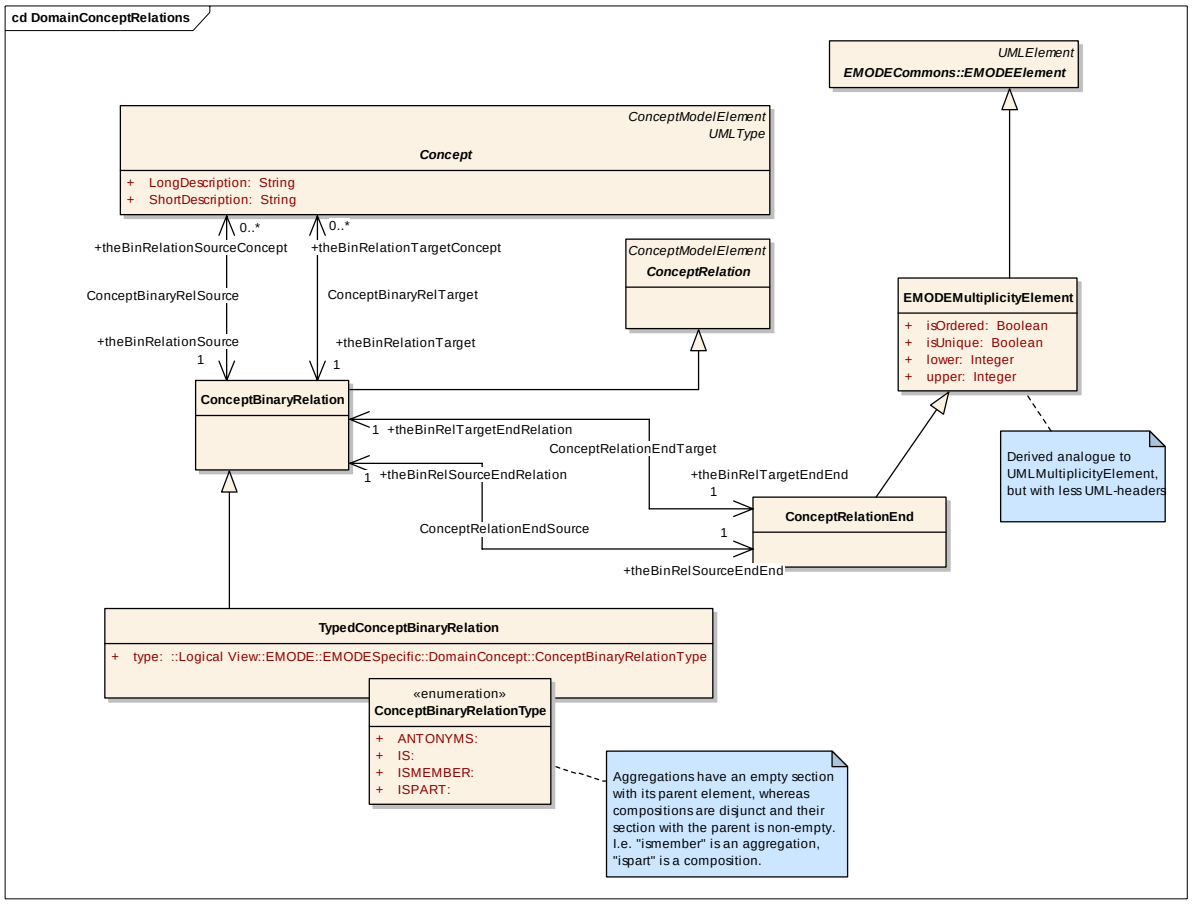
Created By: Andreas Petter on 24.03.2006
Last Modified: 19.06.2006
Version: 1.0. False
GUID: {F1766269-D048-43f6-B44F-C970009F3CDB}

The domain concept model is the model that describes the relations between all possible entities. A concept - which is the EMODE name for entity - has an associated type and may or may not be abstract or persitent.



DomainConceptRelations - *(Class diagram)*

Created By: Alexander Behring on 29.03.2006
Last Modified: 10.07.2006
Version: 1.0. False
GUID: {1477B2EF-FD93-45f9-81C5-F4BFA9D201D4}



AddTypeToArrayOperation

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.24.03.2006.
GUID: {453C96F6-A20A-41f1-B50D-55B26A4EA76B}

This operations adds another type to a structure.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association	Public	Public	Adds another type to the

Connector	Source	Target	Notes
ArrayAddSource Bi-Directional	theStructureToAdd To Structure	theOperationForAd ding AddTypeToArrayO peration	structrue
Association ArrayAddTarget Bi-Directional	Public theArrayAddOperat ion AddTypeToArrayO peration	Public theTypeToAddToA nArray DataType	Adds another type to the structure.

BaseClass

Type: **Class DataType**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.29.03.2006.
GUID: {99BD270D-813E-407e-A2D7-190E8BE9D653}

This class is a placeholder for all the base classes (e.g. Integer, Double, Float, String, etc.) that may contain basic elements.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public BaseClass	Public DataType	

Concept

Type: **Class ConceptModelElement, UMLType**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.24.03.2006.

GUID: {F6B30081-D73B-472d-B4FA-E6D03293F84D}

A concept is a representation of an entity.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public Concept	Public UMLType	
Association ConceptTypeRelation Bi-Directional	Public theConceptToType Concept	Public theTypeOfTheConcept DataType	A type may be associated to any number of concepts, which may have one type, each.
Association ConceptBinaryRelSource Bi-Directional	Public theBinRelationSourceConcept Concept	Public theBinRelationSource ConceptBinaryRelation	A concept may be related to other concepts.
Association ConceptBinaryRelTarget Bi-Directional	Public theBinRelationTargetConcept Concept	Public theBinRelationTarget ConceptBinaryRelation	A concept may be related to another concept.
Generalization Source -> Destination	Public Concept	Public ConceptModelElement	

Attributes

Attribute	Notes	Constraints and tags

Attribute	Notes	Constraints and tags
LongDescription String Public	Describes the concept in an extensive way as a detailed information for the user	*Default:*
ShortDescription String Public	Describes the concept in an short way as a hint for the user	*Default:*

ConceptBinaryRelation

Type: **Class ConceptRelation**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 28.03.2006. *Last modified on*.18.04.2006.
GUID: {7D364CFC-E707-4a21-8A84-52AECC930683}

A BinaryRelation is a relation that connects two concepts

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association Source -> Destination	Public ConceptBinaryRelation	Public ConceptBinaryRelation	
Association ConceptBinaryRelSource Bi-Directional	Public theBinRelationSourceConcept Concept	Public theBinRelationSource ConceptBinaryRelation	A concept may be related to other concepts.

Connector	Source	Target	Notes
Association ConceptBinaryRelTarget Bi-Directional	Public theBinRelationTargetConcept Concept	Public theBinRelationTarget ConceptBinaryRelation	A concept may be related to another concept.
Generalization Source -> Destination	Public ConceptBinaryRelation	Public ConceptRelation	
Association ConceptRelationEndSource Bi-Directional	Public theBinRelSourceEndEnd ConceptRelationEnd	Public theBinRelSourceEndRelation ConceptBinaryRelation	The source end of a concept-concept relation
Generalization Source -> Destination	Public TypedConceptBinaryRelation	Public ConceptBinaryRelation	
Association ConceptRelationEndTarget Bi-Directional	Public theBinRelTargetEndEnd ConceptRelationEnd	Public theBinRelTargetEndRelation ConceptBinaryRelation	The description of the target end in a concept-concept relation

ConceptBinaryRelationType

Type: **Enumeration**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.30.03.2006.
GUID: {5E73640A-3792-4405-A51E-4F1EC6BC8CC5}

Several flags which explain how two concepts are related.

Custom Properties

- isActive =

Connections

Connector	Source	Target	Notes
NoteLink Source -> Destination	Public Note	Public ConceptBinaryRela	

Connector	Source	Target	Notes
		tionType	

Attributes

Attribute	Notes	Constraints and tags
ANTONYMS Public		*Default:*
IS Public		*Default:*
ISMEMBER Public		*Default:*
ISPART Public		*Default:*

ConceptModelElement

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 18.04.2006. *Last modified on*.18.04.2006.
GUID: {0AFFEED0-B5A7-4d24-B95C-7E146E6CF10D}

An element of the DomainConcept model

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public Concept	Public ConceptModelElement	
Generalization Source -> Destination	Public ConceptRelation	Public ConceptModelElement	
Generalization Source -> Destination	Public ConceptModelElement	Public EMODENamedElement	

Attributes

Attribute	Notes	Constraints and tags
persistency PersistentTransientFlaglist Public	Defines whether the concept model element is being made persistent	*Default:*

ConceptRelation

Type: **Class ConceptModelElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 18.04.2006. *Last modified on*.18.04.2006.
GUID: {92BCE036-3D17-44db-99B7-D5CC8909DD73}

A relation between concepts

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ConceptBinaryRelation	Public ConceptRelation	
Generalization Source -> Destination	Public ConceptRelation	Public ConceptModelElement	

ConceptRelationEnd

Type: **Class EMODEMultiplicityElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 30.03.2006. *Last modified on*.30.03.2006.
GUID: {34988B36-CBFC-4faa-B7C6-47CA5BB81B65}

Describes an end of a Relation between concepts

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ConceptRelationEnd	Public EMODEMultiplicityElement	
Association ConceptRelationEnd Source Bi-Directional	Public theBinRelSourceEndEnd ConceptRelationEnd	Public theBinRelSourceEndRelation ConceptBinaryRelation	The source end of a concept-concept relation
Association ConceptRelationEnd	Public	Public	The description of the target

Connector	Source	Target	Notes
Target Bi-Directional	theBinRelTargetEndEnd ConceptRelationEnd	theBinRelTargetEndRelation ConceptBinaryRelation	end in a concept-concept relation

DataType

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.24.03.2006.
GUID: {38DD3322-16F1-4349-A927-CA1EC48DEA58}

A type is an abstract class, that contains a variable to store values.

Custom Properties

- isActive =

Tagged Values

- isAbstract = true.

Connections

Connector	Source	Target	Notes
Association ConceptTypeRelation Bi-Directional	Public theConceptToTypeConcept	Public theTypeOfTheConcept DataType	A type may be associated to any number of concepts, which may have one type, each.
Generalization Source -> Destination	Public BaseClass	Public DataType	
Generalization Source -> Destination	Public Structure	Public DataType	
Association ArrayAddTarget Bi-Directional	Public theArrayAddOperation AddTypeToArrayOperation	Public theTypeToAddToAnArray DataType	Adds another type to the structure.
Generalization Source -> Destination	Public DataType	Public EMODENamedEle	

Connector	Source	Target	Notes
		ment	

Attributes

Attribute	Notes	Constraints and tags
value string Public	Contains the value.	*Default:* [isStatic = false]

EMODEMultiplicityElement

Type: **Class EMODEElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 30.03.2006. *Last modified on*.30.03.2006.
GUID: {0575F690-0844-4d0b-80E1-44BCBB599E94}

An element which can have a multiplicity

Custom Properties

- isActive =

Tagged Values

- isAbstract = true.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ConceptRelationEnd	Public EMODEMultiplicityElement	
NoteLink Source -> Destination	Public Note	Public EMODEMultiplicityElement	
Generalization Source -> Destination	Public EMODEMultiplicityElement	Public EMODEElement	

Connector	Source	Target	Notes

Attributes

Attribute	Notes	Constraints and tags
isOrdered Boolean Public		*Default:* [isStatic = false]
isUnique Boolean Public		*Default:* [isStatic = false]
lower Integer Public	The lower bound of the cardinality - a non-negative integer	*Default:* [isStatic = false]
upper Integer Public	The upper bound of the cardinality - a non-negative integer bigger or equal to "lower" or -1 for unlimited	*Default:* [isStatic = false]

PersistentTransientFlaglist

Type: **Enumeration**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.24.03.2006.
GUID: {D56B86A7-ED7A-40c1-9720-91BBB2E2F01A}

This represents the flags that may be used to identify whether a concept is made persitent.

Custom Properties

- isActive =

Attributes

Attribute	Notes	Constraints and tags
PERSISTENT Public		*Default:*
TRANSIENT Public		*Default:*

PersistentTransientType

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.24.03.2006.
GUID: {ACD96EBD-94C6-439c-8B18-6C0146177034}

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Structure

Type: **Class DataType**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.24.03.2006.
GUID: {33232812-4ABD-4dfd-B887-E9DC00B6E517}

This resembles some sort of Type structrue. It may be seen as an associative class type.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public Structure	Public DataType	
Association ArrayAddSource Bi-Directional	Public theStructureToAddTo Structure	Public theOperationForAdding AddTypeToArrayOperation	Adds another type to the structrue

TypedConceptBinaryRelation

Type: **Class ConceptBinaryRelation**
Status: Proposed. Version 1.0. Phase 1.0.
Package: DomainConcept *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.30.03.2006.
GUID: {3B9FF678-44B9-4a9a-A049-F8F9C94C9813}

A relation between 2 concepts. The relation describes how two concepts are related and how they should be used. Using this relation one may build something like an UML class diagram based ontology.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes

Connector	Source	Target	Notes
Generalization Source -> Destination	Public TypedConceptBinaryRelation	Public ConceptBinaryRelation	

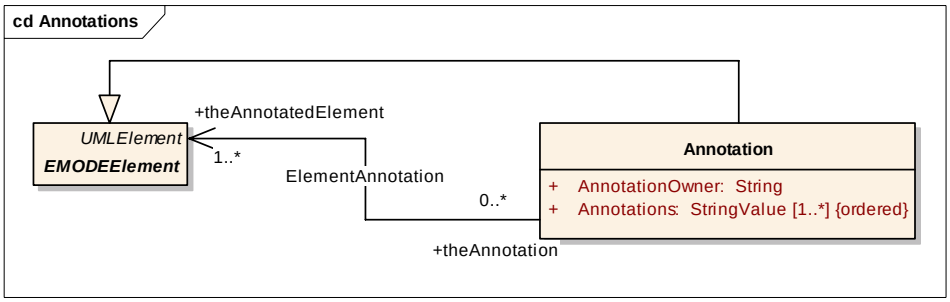
Attributes

Attribute	Notes	Constraints and tags
type ::Logical View::EMODE::EMODESpecific::DomainConcept::ConceptBinaryRelationType Public	The type of the relation (e.g. whether a concept may manipulate another one).	*Default:* [isStatic = false]

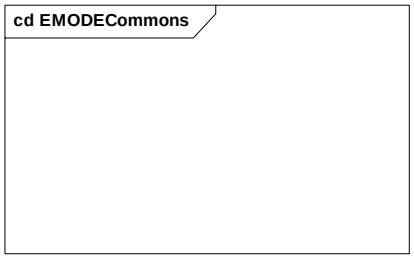
EMODECommons

Type: **Package**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODESpecific
Detail: *Created on* 13.04.2006 10:57:39. *Last modified on* 13.04.2006 10:58:45
GUID: {AEC46739-FFA1-4cd7-8B18-D6E9126A471B}

Annotations - *(Class diagram)*
Created By: Alexander Behring on 27.04.2006
Last Modified: 29.06.2006
Version: 1.0. False
GUID: {0F7E953F-A6AB-4c29-BD6E-581E34651D51}

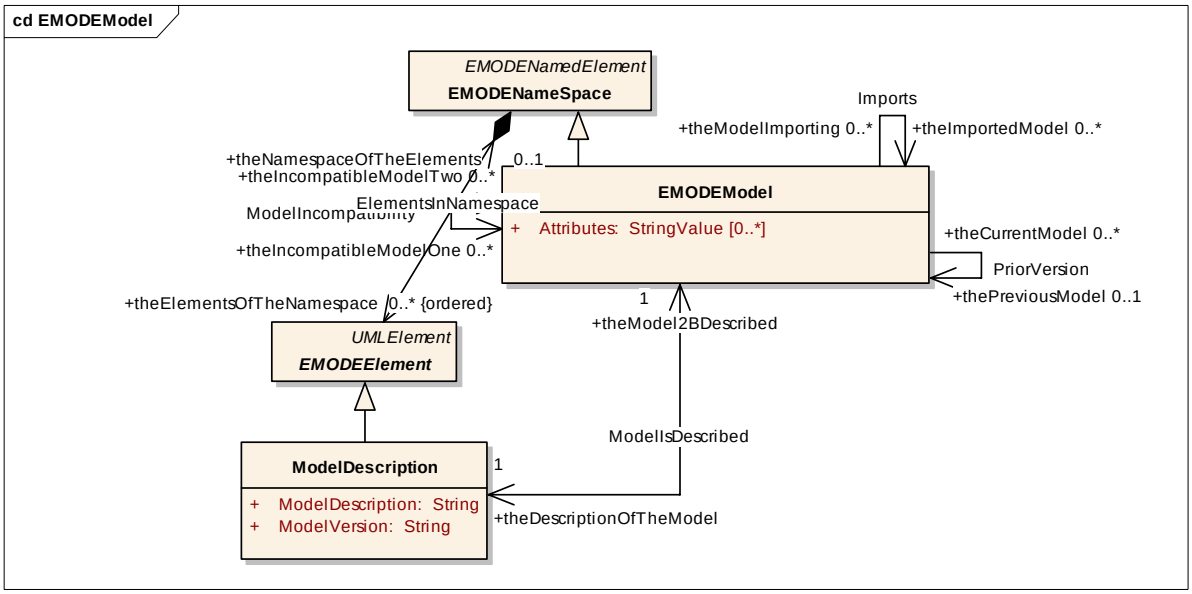


EMODECommons - *(Class diagram)*
Created By: Alexander Behring on 13.04.2006
Last Modified: 21.06.2006
Version: 1.0. False
GUID: {DAA40ED5-8208-49b1-AA48-6FFF7778D213}



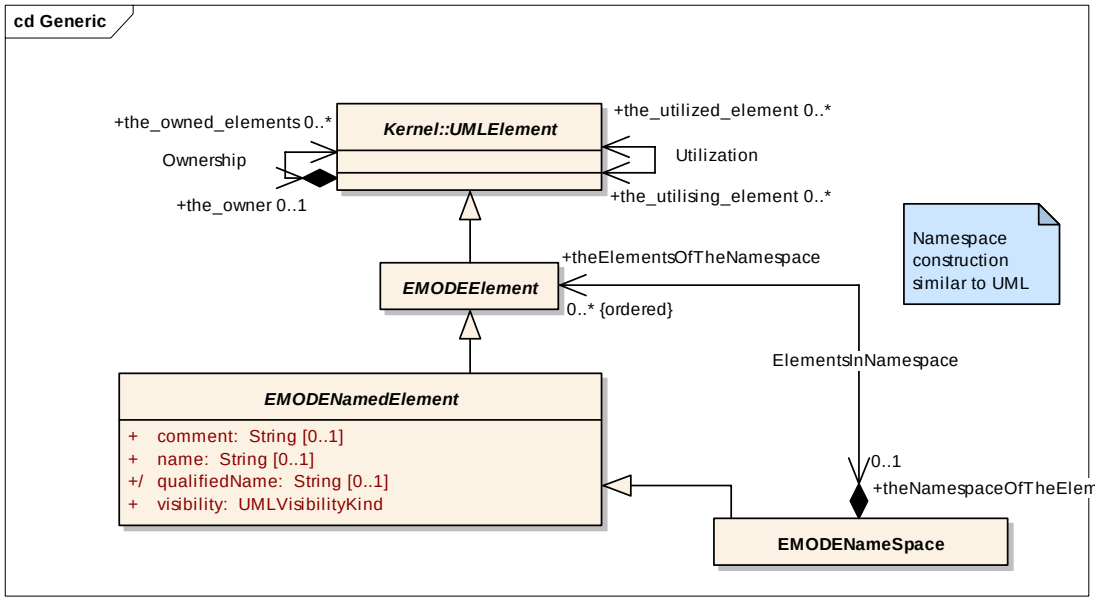
EMODEModel - *(Class diagram)*

Created By: Alexander Behring on 21.06.2006
Last Modified: 21.06.2006
Version: 1.0. False
GUID: {BCA46387-703D-4b45-87F4-AB6F39F06FB0}



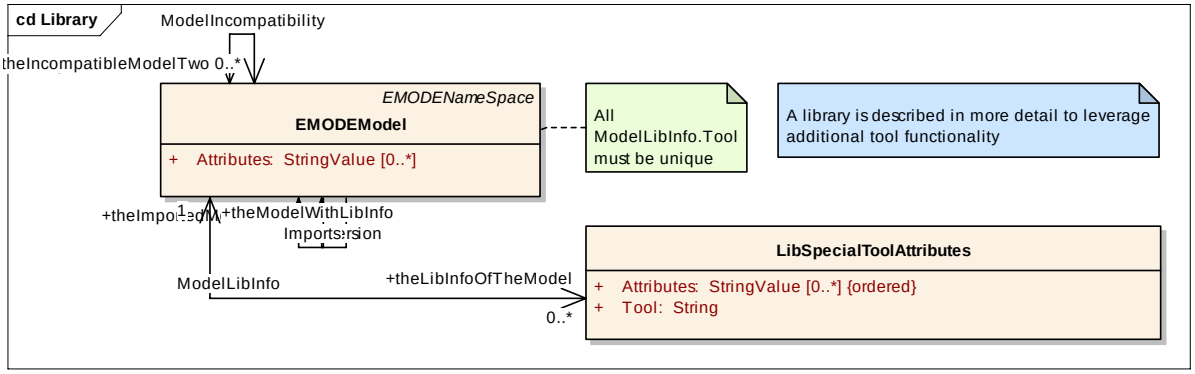
Generic - *(Class diagram)*

Created By: Alexander Behring on 16.06.2006
Last Modified: 21.06.2006
Version: 1.0. False
GUID: {232BE106-0DFB-4d31-BEE6-206135B8669B}



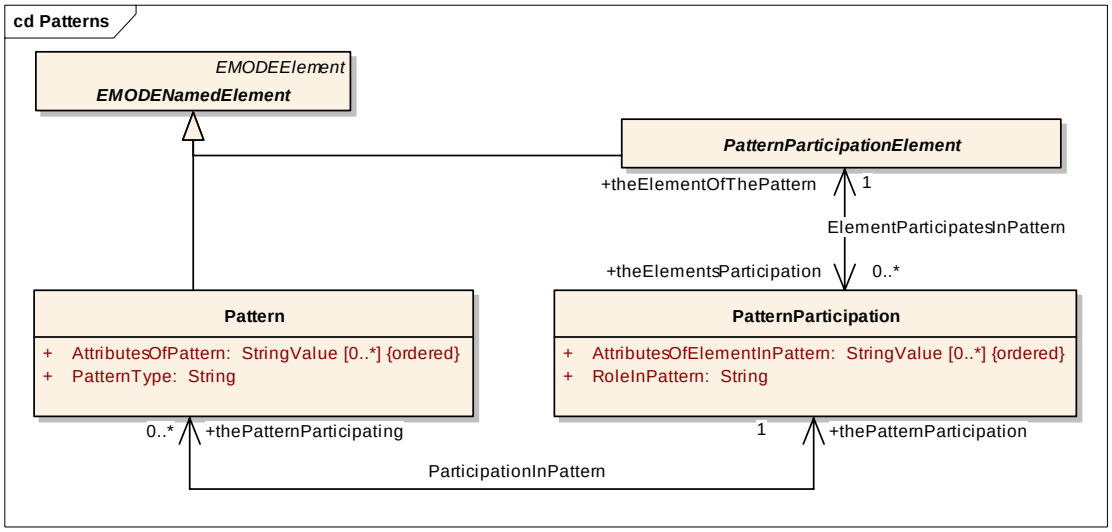
Library - *(Class diagram)*

Created By: Alexander Behring on 21.04.2006
Last Modified: 21.06.2006
Version: 1.0. False
GUID: {F036633D-3391-4906-9E5A-D66D2A58B64A}



Patterns - *(Class diagram)*

Created By: Alexander Behring on 13.04.2006
Last Modified: 20.06.2006
Version: 1.0. False
GUID: {0F179E15-1480-4cdf-BC98-9A2AA332EA9C}



Annotation

Type: **Class EMODEElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODECommons *Keywords:*
Detail: *Created on* 27.04.2006. *Last modified on*.29.06.2006.
GUID: {0474837F-1A89-4df7-AAA0-7ABB24EFB579}

An annotation of some party to an element, it could contain things like layout infos, editor states, documentation, ...

Annotations that are used to parameterize transformations are also written in this element. Two marking schemes are directly at hand:
- The AnnotationOwner is set to the transformation name, with a prefix like "trafo_"
- The AnnotationOwner is set to "Transformation" and the StringValue name is set to a (within the domain of transformations) unique key.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public Annotation	Public EMODEElement	
Association	Public	Public	Connects an annotation with

Connector	Source	Target	Notes
ElementAnnotation Source -> Destination	theAnnotation Annotation	theAnnotatedElement EMODEElement	an EMODEElement

Attributes

Attribute	Notes	Constraints and tags
AnnotationOwner String Public	The owner of the annotation	*Default:*
Annotations StringValue Public [1..*]	The annotations of the owner	*Default:*

EMODEElement

Type: **Class UMLElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODECommons *Keywords:*
Detail: *Created on* 16.06.2006. *Last modified on*.16.06.2006.
GUID: {3A206D05-F27F-4d45-8336-13CEABBFD051}

An element used by EMODE.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ModalityRestriction Property	Public EMODEElement	

Connector	Source	Target	Notes
Generalization Source -> Destination	Public EMODEMultiplicityElement	Public EMODEElement	
Generalization Source -> Destination	Public Annotation	Public EMODEElement	
Aggregation ElementsInNamespace Bi-Directional	Public theElementsOfTheNamespace EMODEElement	Public theNamespaceOfTheElements EMODENameSpace	Associates named elements to a namespace
Association ElementAnnotation Source -> Destination	Public theAnnotation Annotation	Public theAnnotatedElement EMODEElement	Connects an annotation with an EMODEElement
Generalization Source -> Destination	Public EMODEElement	Public UMLElement	
Generalization Source -> Destination	Public EMODENamedElement	Public EMODEElement	
Generalization Source -> Destination	Public ModelDescription	Public EMODEElement	

EMODEModel

Type: **Class EMODENameSpace**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODECommons *Keywords:*
Detail: *Created on* 21.06.2006. *Last modified on*.21.06.2006.
GUID: {F4722F14-A63C-4248-981C-327543C9BB3B}

An EMODEModel, i.e. an instance of the meta model.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public EMODEModel	Public EMODENameSpace	
Association ModelIsDescribed Bi-Directional	Public theModel2BDescribed EMODEModel	Public theDescriptionOfTheModel ModelDescription	
Association ModelLibInfo Bi-Directional	Public theLibInfoOfTheModel LibSpecialToolAttributes	Public theModelWithLibInfo EMODEModel	
Association Imports Source -> Destination	Public theModelImporting EMODEModel	Public theImportedModel EMODEModel	A model uses another model
Association PriorVersion Source -> Destination	Public theCurrentModel EMODEModel	Public thePreviousModel EMODEModel	
Association ModelIncompatibility Bi-Directional	Public theIncompatibleModelOne EMODEModel	Public theIncompatibleModelTwo EMODEModel	
NoteLink Source -> Destination	Public Note	Public EMODEModel	

Attributes

Attribute	Notes	Constraints and tags
Attributes StringValue Public [0..*]	Developer defined attributes	*Default:*

EMODENamedElement

Type: **Class EMODEElement**
Status: Proposed. Version . Phase .
Package: EMODECommons *Keywords:*
Detail: *Created on* 02.09.2005. *Last modified on*.16.06.2006.
GUID: {032E8D38-446A-46ce-AD62-8E6031A20600}

An EMODENamedElement represents elements that may have a name. The name is used for identification of the named element within the namespace in which it is defined. An UMLNamedElement also has a qualified name that allows it to be unambiguously identified within a hierarchy of nested namespaces. EMODENamedElement is an abstract metaclass.

Custom Properties

- isActive =

Tagged Values

- isAbstract = true.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public Task2GoalRelation	Public EMODENamedElement	
Generalization Source -> Destination	Public FCACall	Public EMODENamedElement	
Generalization	Public UMLType	Public EMODENamedElement	
Generalization Source -> Destination	Public AUIComponentRelation	Public EMODENamedElement	
Generalization Source -> Destination	Public DataType	Public EMODENamedElement	
Generalization Source -> Destination	Public Service	Public EMODENamedElement	
Generalization Source -> Destination	Public Deviceport	Public EMODENamedEle	

Connector	Source	Target	Notes
		ment	
Generalization Source -> Destination	Public ServiceParameter	Public EMODENamedElement	
Generalization Source -> Destination	Public Goal	Public EMODENamedElement	
Generalization Source -> Destination	Public ContextProvider	Public EMODENamedElement	
Generalization Source -> Destination	Public TaskRelation	Public EMODENamedElement	
Generalization Source -> Destination	Public EMODEInteractionFlag	Public EMODENamedElement	
Generalization Source -> Destination	Public ContextDataUse	Public EMODENamedElement	
Generalization Source -> Destination	Public ModalityRestrictionProfile	Public EMODENamedElement	
Generalization Source -> Destination	Public EMODENameSpace	Public EMODENamedElement	
Generalization Source -> Destination	Public Pattern	Public EMODENamedElement	
Generalization Source -> Destination	Public ConceptModelElement	Public EMODENamedElement	
Generalization Source -> Destination	Public EMODENamedElement	Public EMODEElement	
Generalization Source -> Destination	Public PatternParticipationElement	Public EMODENamedElement	

Connector	Source	Target	Notes
Generalization Source -> Destination	Public TaskPartition	Public EMODENamedElement	
Generalization Source -> Destination	Public ContextQueryOutputPin	Public EMODENamedElement	
Generalization Source -> Destination	Public FCACallParameter	Public EMODENamedElement	
Generalization Source -> Destination	Public FCACallResult	Public EMODENamedElement	

Attributes

Attribute	Notes	Constraints and tags
comment String Public [0..1]	The property contains the comments which are added to the namedElement.	*Default:* [isStatic = false]
name String Public [0..1]	The property contains the name of the element. The elements name must be unique in the namespace.	*Default:* [isStatic = false]
qualifiedName String Public [0..1]	The derived attribute returns the qualified name of the NamedElement as a list of string.	*Default:* [isStatic = false]
visibility UMLVisibilityKind Public	The property contains the visibility of the element in its namespace.	*Default:* [isStatic = false]

EMODENameSpace

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODECommons *Keywords:*
Detail: *Created on* 21.06.2006. *Last modified on*.21.06.2006.
GUID: {6D5481FA-5348-467b-BE42-7CFBDF3D40BB}

A namespace can contain named elements. Since namespaces are named elements, too, they can be nested.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public EMODENameSpace	Public EMODENamedElement	
Aggregation ElementsInNamespace Bi-Directional	Public theElementsOfTheNamespace EMODEElement	Public theNamespaceOfTheElements EMODENameSpace	Associates named elements to a namespace
Generalization Source -> Destination	Public EMODEModel	Public EMODENameSpace	

LibSpecialToolAttributes

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODECommons *Keywords:*
Detail: *Created on* 21.04.2006. *Last modified on*.21.06.2006.
GUID: {CD2AECF9-5991-4db0-900C-B399D6431732}

Attributes of the library which are specific to a certein tool and not covered by the general attributes

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association ModelLibInfo Bi-Directional	Public theLibInfoOfTheModel LibSpecialToolAttributes	Public theModelWithLibInfo EMODEModel	

Attributes

Attribute	Notes	Constraints and tags
Attributes StringValue Public [0..*]	The attributes	*Default:* [isStatic = false]
Tool String Public	The identifier of the tool	*Default:* [isStatic = false]

ModelDescription

Type: **Class EMODEElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODECommons *Keywords:*
Detail: *Created on* 21.04.2006. *Last modified on*.21.06.2006.
GUID: {C9315464-FBEF-4d76-B591-CA8BCFDB051B}

Description of the library

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ModelDescription	Public EMODEElement	
Association ModelIsDescribed Bi-Directional	Public theModel2BDescribed EMODEModel	Public theDescriptionOfTheModel ModelDescription	

Attributes

Attribute	Notes	Constraints and tags
ModelDescription String Public	A description of what this model is used for	*Default:* [isStatic = false]
ModelVersion String Public	Version of the model in the format x.x.x	*Default:* [isStatic = false]

Pattern

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODECommons *Keywords:*
Detail: *Created on* 13.04.2006. *Last modified on*.13.04.2006.
GUID: {4E0FF8F6-5381-4fa3-8B45-2693448B26BF}

Describes a pattern, with the pattern wide information. All detailed information about participating elements' attributes is in the respective PatternParticipation elements with these elements.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public Pattern	Public EMODENamedElement	
Association ParticipationInPattern Bi-Directional	Public thePatternParticipation PatternParticipation	Public thePatternParticipating Pattern	

Attributes

Attribute	Notes	Constraints and tags
AttributesOfPattern StringValue Public [0..*]	The attributes of this pattern	*Default:* [isStatic = false]
PatternType String Public	The type of pattern which the element participates in	*Default:* [isStatic = false]

PatternParticipation

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODECommons *Keywords:*

Detail: *Created on* 13.04.2006. *Last modified on*.13.04.2006.
GUID: {A71DF7B6-C7D0-4368-BDED-52027B52915E}

Describes the participation of an element in a pattern

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association ElementParticipatesInPattern Bi-Directional	Public theElementOfThePattern PatternParticipationElement	Public theElementsParticipation PatternParticipation	Elements derived from PatternParticitpationElement can participate in 0..* patterns via a PatternParticipation
Association ParticipationInPattern Bi-Directional	Public thePatternParticipation PatternParticipation	Public thePatternParticipating Pattern	

Attributes

Attribute	Notes	Constraints and tags
AttributesOfElementInPattern StringValue Public [0..*]	The attributes of this element within the pattern it participates in	*Default:* [isStatic = false]
RoleInPattern String Public	The role of this element wihtin the pattern	*Default:* [isStatic = false]

PatternParticipationElement

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODECommons *Keywords:*
Detail: *Created on* 13.04.2006. *Last modified on*.13.04.2006.
GUID: {94F9E3EB-7B1A-4ba9-A082-EE8901B835CD}

An element derived from this class may participate in a pattern

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

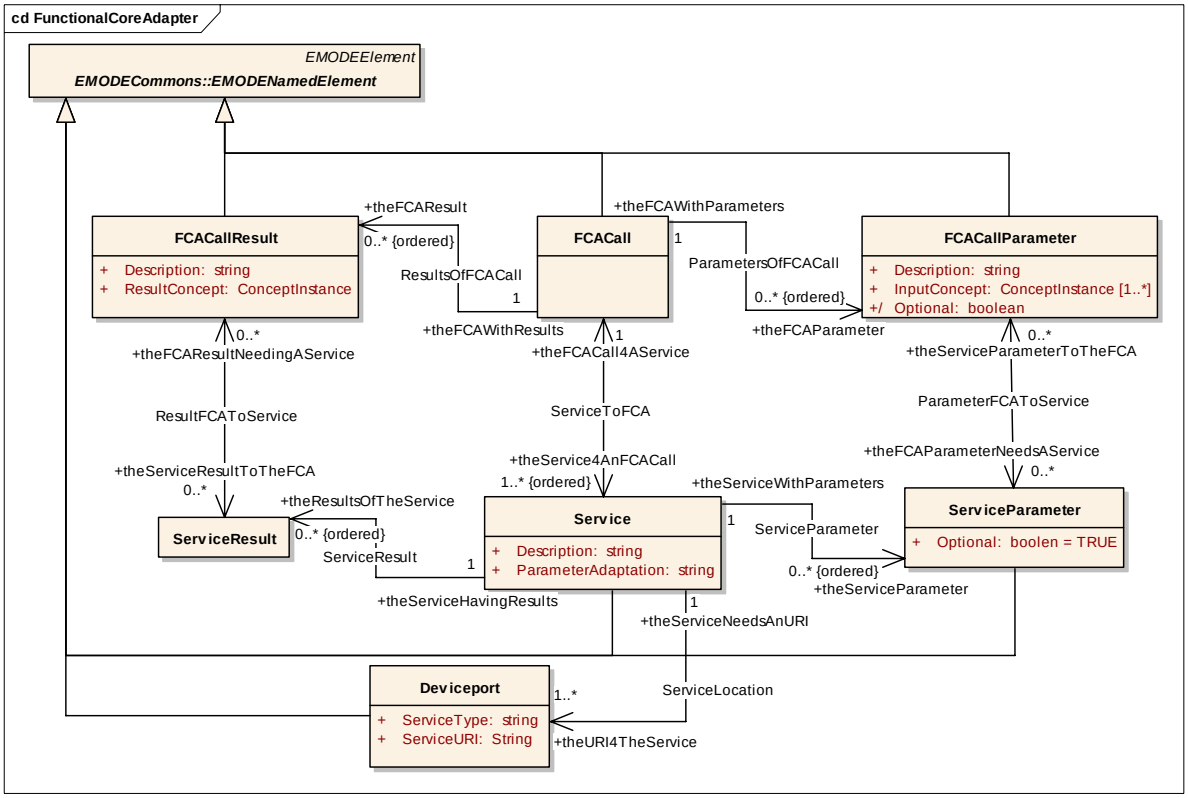
Connector	Source	Target	Notes
Generalization Source -> Destination	Public TaskNode	Public PatternParticipation Element	
Generalization Source -> Destination	Public TaskEdge	Public PatternParticipation Element	
Generalization Source -> Destination	Public AUIComponent	Public PatternParticipation Element	
Association ElementParticipatesIn Pattern Bi-Directional	Public theElementOfThePattern PatternParticipation Element	Public theElementsParticipation PatternParticipation	Elements derived from PatternParticitpationElement can participate in 0..* patterns via a PatternParticipation
Generalization Source -> Destination	Public PatternParticipation Element	Public EMODENamedElement	

FunctionalCoreAdapter

Type: **Package**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODESpecific
Detail: *Created on* 24.03.2006 14:57:11. *Last modified on* 02.06.2006 09:36:06

GUID: {FC3A8A8D-96E7-468d-AC89-7E50489207D1}

FunctionalCoreAdapter - *(Class diagram)*
Created By: Alexander Behring on 24.03.2006
Last Modified: 23.07.2006
Version: 1.0. False
GUID: {CDD2433D-412E-4c9a-9A82-6BF3D815134E}



Deviceport

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: FunctionalCoreAdapter *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.24.03.2006.
GUID: {7BB9EBE5-CD49-41d3-AE2F-5FC8A68618DA}

A deviceport is a device description that has implementations for the associated services.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public Deviceport	Public EMODENamedElement	
Association ServiceLocation Source -> Destination	Public theServiceNeedsAnURI Service	Public theURI4TheService Deviceport	

Attributes

Attribute	Notes	Constraints and tags
ServiceType string Public	The type of the service (e.g., local, remote per SOAP, etc.)	*Default:*
ServiceURI String Public	The location of the service	*Default:*

FCACall

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: FunctionalCoreAdapter *Keywords:*
Detail: *Created on* 03.06.2006. *Last modified on*.03.06.2006.
GUID: {DD859D48-CD6C-44c3-B288-251E86880B28}

The interface to other elements, where an FCA is defined

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public FCACall	Public EMODENamedElement	
Association TaskIsExecutedBy Bi-Directional	Public theTaskNode2BeExec TaskNode	Public theFCACallToExecATaskNode FCACall	The connection between a system task and a FCA call
Association ServiceToFCA Bi-Directional	Public theFCACall4AService FCACall	Public theService4AnFCACall Service	Several services can be used to realize one FCA call. The results and parameters must be the same.
Association ParametersOfFCACall Destination -> Source	Public theFCAParameter FCACallParameter	Public theFCAWithParameters FCACall	The parameters associated to an FCA call. They must/can be filled by the calling entity.
Association ResultsOfFCACall Source -> Destination	Public theFCAWithResults FCACall	Public theFCAResult FCACallResult	The results delivered by an FCA Call. They can be used by the calling entities.

FCACallParameter

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: FunctionalCoreAdapter *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.24.06.2006.
GUID: {3A017190-7555-46be-ADDF-C25AD453BCE0}

A parameter is a parameter that may be given to a service. As one may see, parameters lack the direction attribute of the original UML specification. We define a parameter inherently as input.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association ParameterFCAToService Bi-Directional	Public theFCAParameterNeedsAService ServiceParameter	Public theServiceParameterToTheFCA FCACallParameter	Connects FCA Call parameters with service parameters
Generalization Source -> Destination	Public FCACallParameter	Public EMODENamedElement	
Association ParametersOfFCACall Destination -> Source	Public theFCAParameter FCACallParameter	Public theFCAWithParameters FCACall	The parameters associated to an FCA call. They must/can be filled by the calling entity.

Attributes

Attribute	Notes	Constraints and tags
Description string Public	Description of the parameter	*Default:*
InputConcept ConceptInstance Public [1..*]	The input concept(s) which can be used in this parameter.	*Default:*
Optional boolean Public	Wheather the parameter is optional. Derived from the connected services.	*Default:*

FCACallResult

Type: **Class EMODENamedElement**

Status: Proposed. Version 1.0. Phase 1.0.
Package: FunctionalCoreAdapter *Keywords:*
Detail: *Created on* 23.06.2006. *Last modified on*.23.06.2006.
GUID: {01236EE1-FC55-4c69-AE56-A400F5DE3C63}

The result of the service

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association ResultFCAToService Bi-Directional	Public theFCAResultNeedingAService FCACallResult	Public theServiceResultToTheFCA ServiceResult	Connects a service result to a FCA result.
Association ResultsOfFCACall Source -> Destination	Public theFCAWithResultsFCACall	Public theFCAResult FCACallResult	The results delivered by an FCA Call. They can be used by the calling entities.
Generalization Source -> Destination	Public FCACallResult	Public EMODENamedElement	

Attributes

Attribute	Notes	Constraints and tags
Description string Public	Description of the result	*Default:*
ResultConcept ConceptInstance Public	The concept delivered by this FCA call	*Default:*

Service

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: FunctionalCoreAdapter *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.24.03.2006.
GUID: {FE31313A-9355-4487-B0FD-89DE7FB61C69}

A service is the prototype function stub of a function implementation.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public Service	Public EMODENamedElement	
Association ServiceResult Source -> Destination	Public theServiceHavingResults Service	Public theResultsOfTheService ServiceResult	An operation may have a result.
Association ServiceParameter Source -> Destination	Public theServiceWithParameters Service	Public theServiceParameter ServiceParameter	An operation may have several parameters.
Association ServiceLocation Source -> Destination	Public theServiceNeedsAnURI Service	Public theURI4TheServiceDeviceport	
Association ServiceToFCA Bi-Directional	Public theFCACall4AService FCACall	Public theService4AnFCACall Service	Several services can be used to realize one FCA call. The results and parameters must be the same.

Attributes

Attribute	Notes	Constraints and tags
Description string Public	A short description of the service	*Default:*
ParameterAdaptation string Public	Describes the adaptation of FCACallParameters to the ServiceParameters in more detail. If given, all ServiceParameters must be explicitly set - the ParameterFCAToService associations then only represent the dependencies.	*Default:*

ServiceParameter

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: FunctionalCoreAdapter *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.24.06.2006.
GUID: {48715C56-AA71-4aa3-9841-A3958EEBA690}

A parameter is a parameter that may be given to a service. As one may see, parameters lack the direction attribute of the original UML specification. This is because we don't want in/out parameters in all kinds of distributed systems - we only provide one result.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association ServiceParameter Source -> Destination	Public theServiceWithParameters Service	Public theServiceParameter ServiceParameter	An operation may have several parameters.
Generalization Source -> Destination	Public ServiceParameter	Public EMODENamedElement	

Connector	Source	Target	Notes
Association ParameterFCAToService Bi-Directional	Public theFCAParameterNeedsAService ServiceParameter	Public theServiceParameterToTheFCA FCACallParameter	Connects FCA Call parameters with service parameters

Attributes

Attribute	Notes	Constraints and tags
Optional boolen Public TRUE	If the service needs this input parameter	*Default:* TRUE

ServiceResult

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: FunctionalCoreAdapter *Keywords:*
Detail: *Created on* 23.06.2006. *Last modified on*.24.06.2006.
GUID: {05B7548E-BAC0-46db-9484-1D2F0A9D7A8B}

The result of the service

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

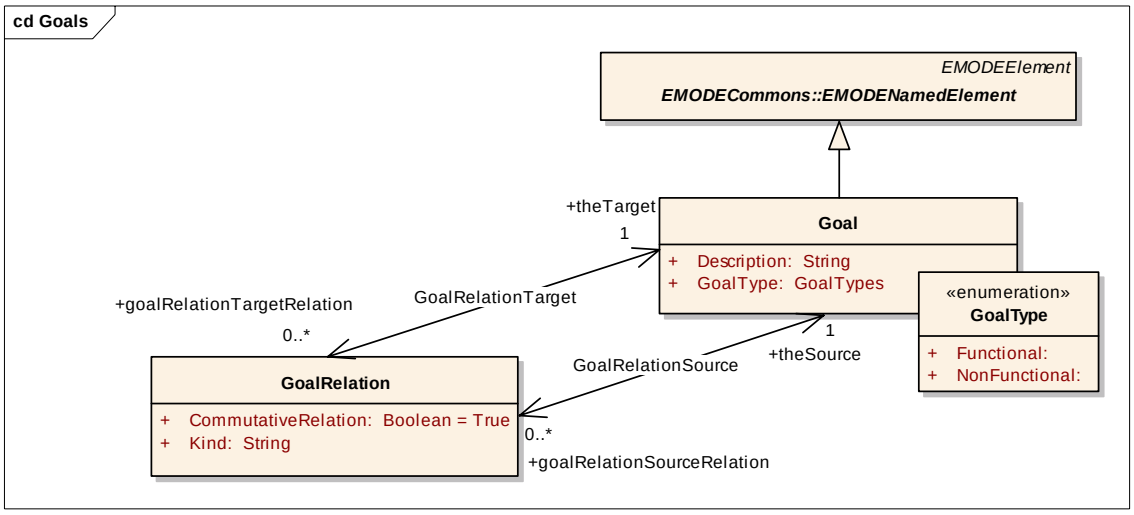
Connector	Source	Target	Notes
Association ResultFCAToService Bi-Directional	Public theFCAResultNeedingAService FCACallResult	Public theServiceResultToTheFCA ServiceResult	Connects a service result to a FCA result.
Association ServiceResult Source -> Destination	Public theServiceHavingR	Public theResultsOfTheSer	An operation may have a result.

Connector	Source	Target	Notes
	esults Service	vice ServiceResult	

Goals

Type: **Package**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODESpecific
Detail: *Created on* 07.03.2006 15:42:53. *Last modified on* 02.06.2006 09:35:33
GUID: {175CFBD8-77AB-49ae-9E85-EAA848294C02}

Goals - *(Class diagram)*
Created By: J. Hößler on 07.03.2006
Last Modified: 16.06.2006
Version: 1.0. False
GUID: {B8327CD1-63F9-42e9-9A75-9175615F1CA2}



Goal

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Goals *Keywords:*
Detail: *Created on* 07.03.2006. *Last modified on*.18.04.2006.
GUID: {5F719533-CEFA-4ce6-8D01-4FF6D2F271CE}

A business goal is a non-technical goal which needs to be achieved

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association GoalsInPartition Bi-Directional	Public theGoalOfThePartition Goal	Public thePartitionOfTheGoal TaskPartition	
Association GoalRelationTarget Bi-Directional	Public goalRelationTargetRelation GoalRelation	Public theTarget Goal	
Association GoalRelationSource Bi-Directional	Public goalRelationSourceRelation GoalRelation	Public theSource Goal	
Generalization Source -> Destination	Public Goal	Public EMODENamedElement	
Association Task2GoalRelationGoal Bi-Directional	Public task2GoalGoal Goal	Public task2GoalGoalAssociation Task2GoalRelation	Connects a Goal to its Association

Attributes

Attribute	Notes	Constraints and tags
Description String Public		*Default:* [isStatic = false]

Attribute	Notes	Constraints and tags
GoalType GoalTypes Public		*Default:*

GoalRelation

Type: **Class**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Goals *Keywords:*
Detail: *Created on* 07.03.2006. *Last modified on*.07.03.2006.
GUID: {ABED00EB-1D02-432f-93AA-BA0BBB18DD16}

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association GoalRelationTarget Bi-Directional	Public goalRelationTarget Relation GoalRelation	Public theTarget Goal	
Association GoalRelationSource Bi-Directional	Public goalRelationSource Relation GoalRelation	Public theSource Goal	

Attributes

Attribute	Notes	Constraints and tags

Attribute	Notes	Constraints and tags
CommutativeRelation Boolean Public True	The relation is commutative	*Default:* True [isStatic = false]
Kind String Public		*Default:* [isStatic = false]

GoalType

Type: **Enumeration**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Goals *Keywords:*
Detail: *Created on* 29.05.2006. *Last modified on*.29.05.2006.
GUID: {E94EB909-732F-45cf-BB5F-90F6174C7A13}

The type of a goal

Custom Properties

- isActive =

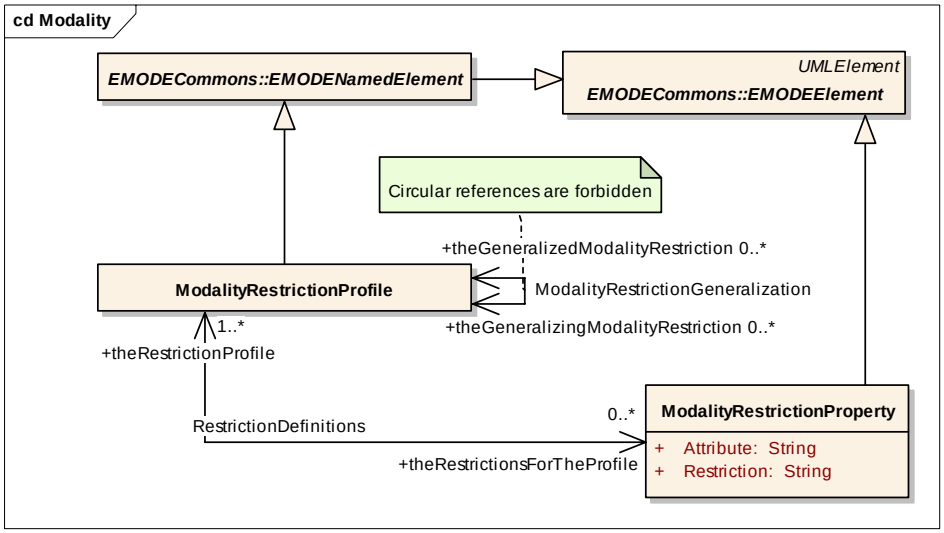
Attributes

Attribute	Notes	Constraints and tags
Functional Public		*Default:*
NonFunctional Public		*Default:*

Modality

Type: **Package**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODESpecific
Detail: *Created on* 07.03.2006 16:57:09. *Last modified on* 08.03.2006 14:11:21
GUID: {0791BA7D-CDA3-4175-9DF3-B06FCCE76DB9}

Modality - *(Class diagram)*
Created By: J. Hößler on 07.03.2006
Last Modified: 21.06.2006
Version: 1.0. False
GUID: {7E7C178B-C658-440a-883F-1620BD697E21}



ModalityRestrictionProfile

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Modality *Keywords:*
Detail: *Created on* 07.03.2006. *Last modified on*.21.06.2006.
GUID: {E19A2BE1-8368-4841-8152-277490A1F018}

Defines restrictions on modality-properties in order to identify a (sub)set of modalities that are adressed by e.g. an AUI

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ModalityRestriction Profile	Public EMODENamedElement	
Association RestrictionDefinitions Bi-Directional	Public theRestrictionProfile ModalityRestriction Profile	Public theRestrictionsForTheProfile ModalityRestriction Property	The restrictions defined for this ModalityRestriction profile
Association ModalityRestrictionGeneralization Bi-Directional	Public theGeneralizedModalityRestriction ModalityRestriction Profile	Public theGeneralizingModalityRestriction ModalityRestriction Profile	Defines that the restriction is generalized by another restriction. I.e. the restrictions defined in the generalizing element are also valid for the generalized element.

ModalityRestrictionProperty

Type: **Class EMODEElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Modality *Keywords:*
Detail: *Created on* 31.03.2006. *Last modified on*.21.06.2006.
GUID: {A6D369EC-0551-43fa-B1D3-FCD2757168EC}

A property which consists of a attribute and a restriction that needs to be applied to it

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes

Connector	Source	Target	Notes
Generalization Source -> Destination	Public ModalityRestriction Property	Public EMODEElement	
Association RestrictionDefinitions Bi-Directional	Public theRestrictionProfile ModalityRestriction Profile	Public theRestrictionsForTheProfile ModalityRestriction Property	The restrictions defined for this ModalityRestriction profile

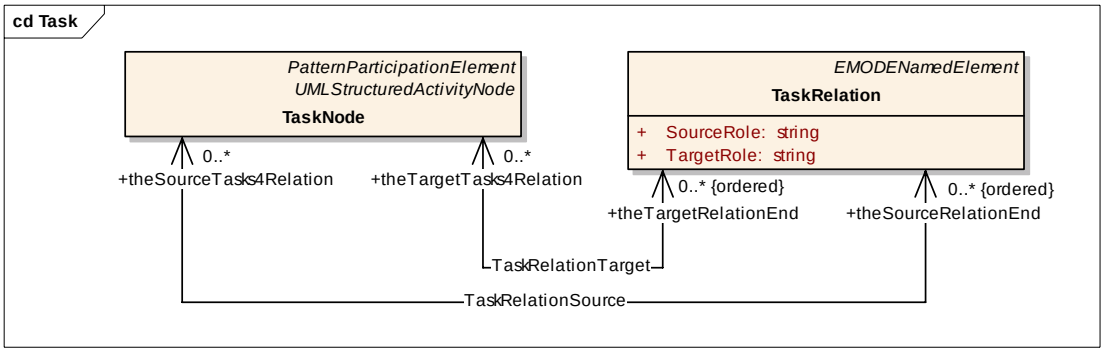
Attributes

Attribute	Notes	Constraints and tags
Attribute String Public	The attribute which the restriction is imposed upon	*Default:*
Restriction String Public	The restriction	*Default:*

Task

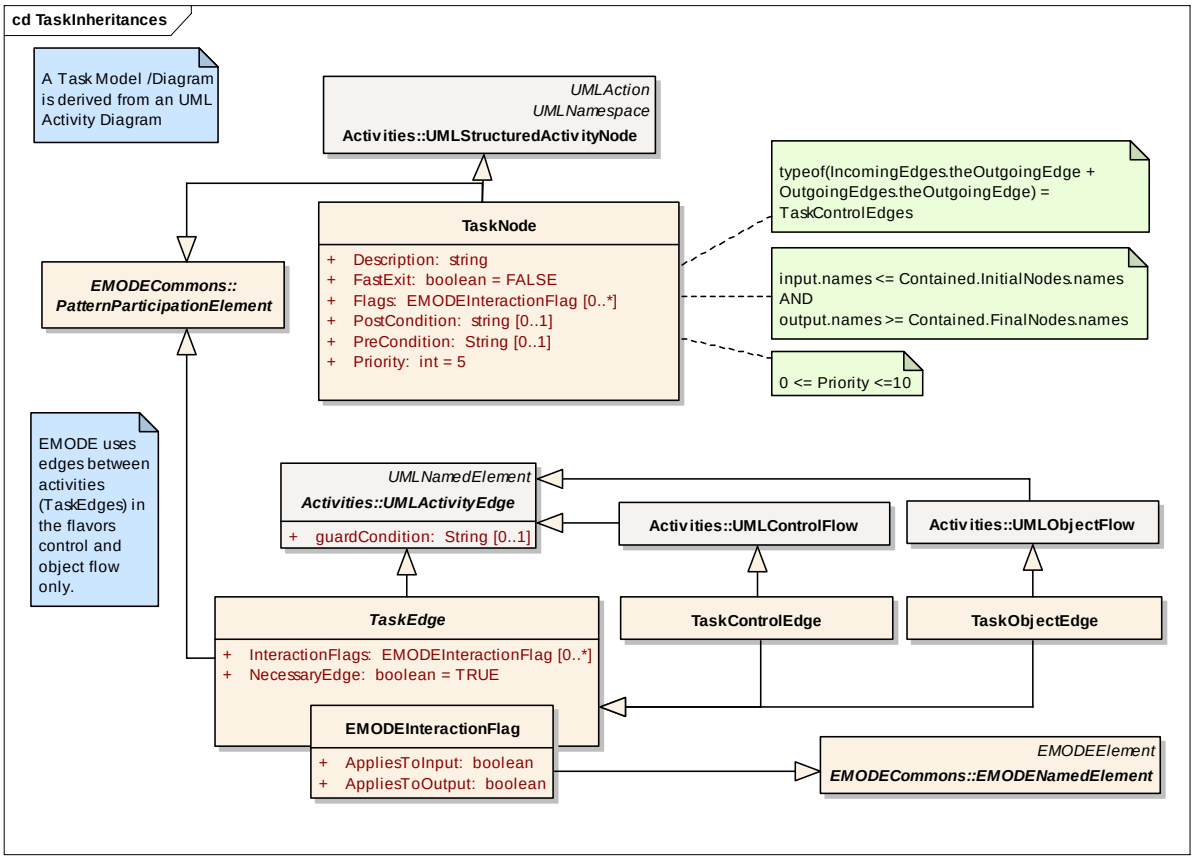
Type: **Package**
Status: Proposed. Version 1.0. Phase 1.0.
Package: EMODESpecific
Detail: *Created on* 07.03.2006 15:58:24. *Last modified on* 24.03.2006 11:13:40
GUID: {A27CD461-F29C-4959-96DA-87D2FC5F0DA6}

Task - *(Class diagram)*
Created By: Alexander Behring on 14.06.2006
Last Modified: 07.07.2006
Version: 1.0. False
GUID: {DB329F1D-BB9E-46e0-9530-2D4139BBE240}



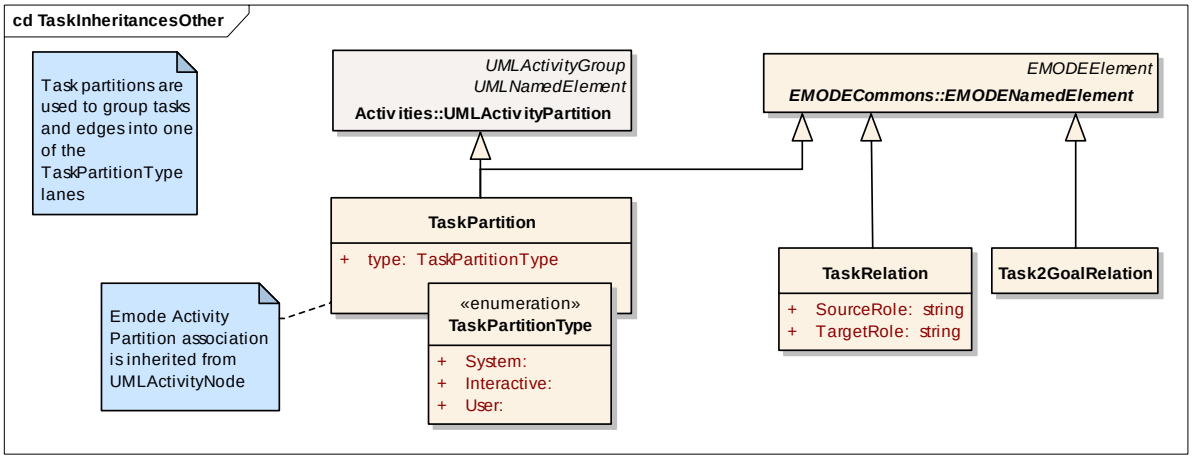
TaskInheritances - *(Class diagram)*

Created By: Alexander Behring on 29.05.2006
Last Modified: 30.06.2006
Version: 1.0. False
GUID: {D7416033-47B3-4391-BED6-C9CA390FD87C}



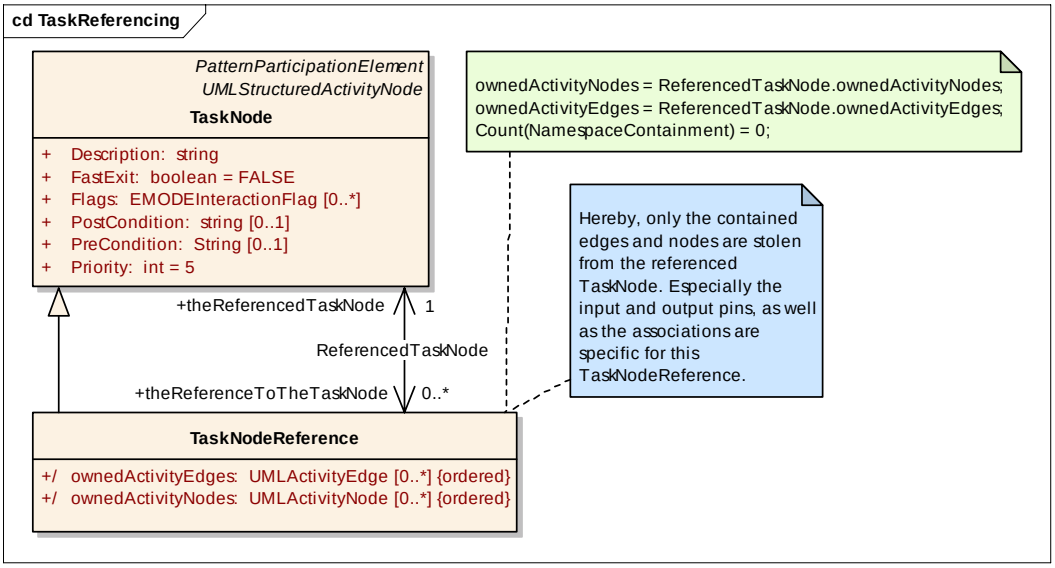
TaskInheritancesOther - *(Class diagram)*

Created By: Alexander Behring on 13.06.2006
Last Modified: 21.06.2006
Version: 1.0. False
GUID: {2786C750-0C82-428d-A63E-5783545F8015}



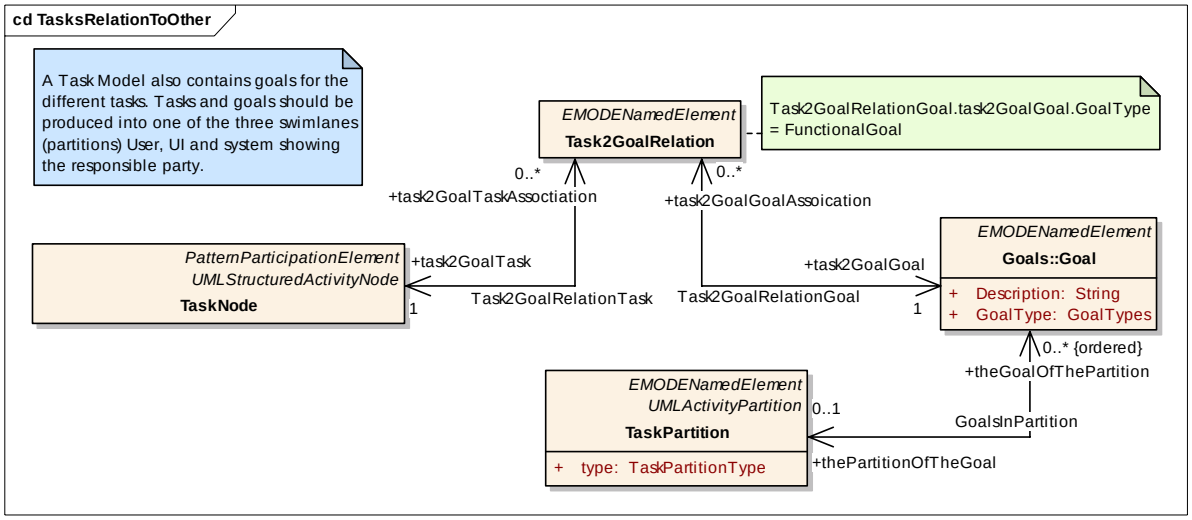
TaskReferencing - *(Class diagram)*

Created By: Alexander Behring on 21.06.2006
Last Modified: 07.07.2006
Version: 1.0. False
GUID: {3714A805-CCCC-478a-9D74-A983897567A2}



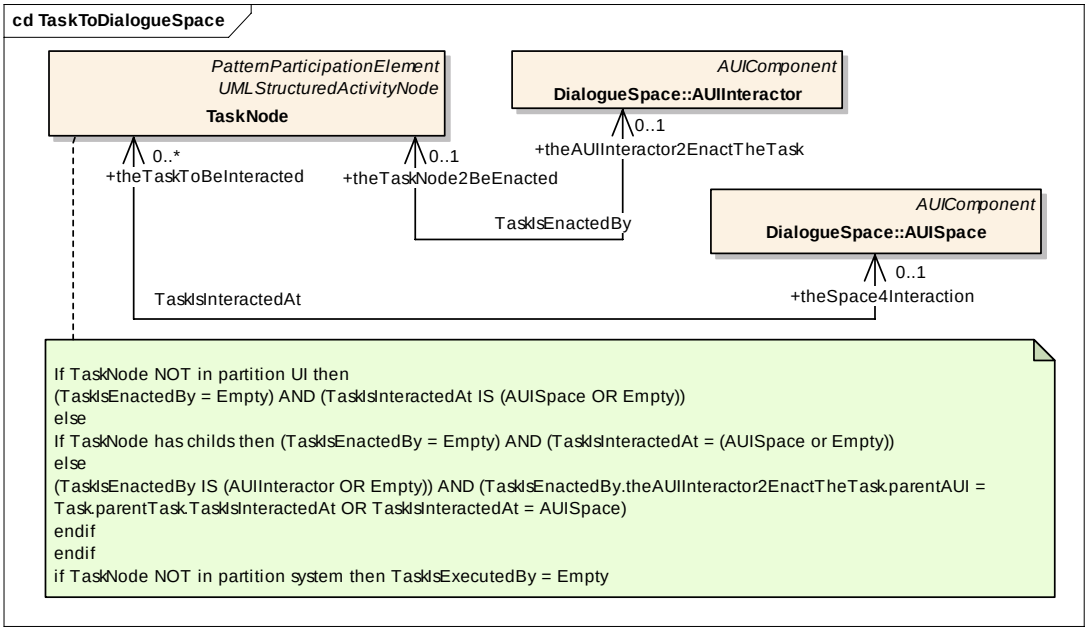
TasksRelationToOther - *(Class diagram)*

Created By: Alexander Behring on 07.03.2006
Last Modified: 07.07.2006
Version: 1.0. False
GUID: {42A498E0-8727-461d-A24D-BA74F7057404}



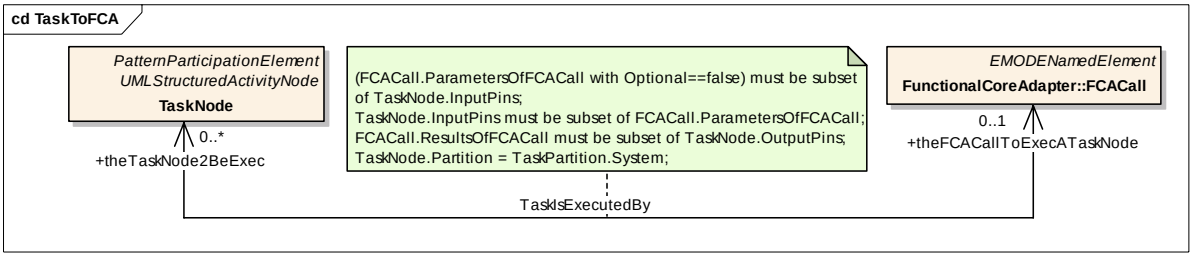
TaskToDialogueSpace - *(Class diagram)*

Created By: Alexander Behring on 30.06.2006
Last Modified: 07.07.2006
Version: 1.0. False
GUID: {99C64DB1-ADAE-4e7b-9BD7-9DE1AD55DE7E}



TaskToFCA - *(Class diagram)*

Created By: Alexander Behring on 30.06.2006
Last Modified: 07.07.2006
Version: 1.0. False
GUID: {532DBDE5-1547-4877-99E7-F9AC7E1215BD}



EMODEInteractionFlag

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Task *Keywords:*
Detail: *Created on* 03.06.2006. *Last modified on*.03.06.2006.
GUID: {F3A986CC-97AB-408e-AF78-5726962E9283}

Flags for giving more information... are user defined

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public EMODEInteraction Flag	Public EMODENamedElement	

Attributes

Attribute	Notes	Constraints and tags
AppliesToInput boolean Public	The flag can be applied to a fusion process	*Default:*

Attribute	Notes	Constraints and tags
AppliesToOutput boolean Public	The flag can be applied to a fission process	*Default:*

Task2GoalRelation

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Task *Keywords:*
Detail: *Created on* 15.03.2006. *Last modified on*.29.05.2006.
GUID: {0F710508-0FF0-496f-8AF4-BDDA10FBC25F}

This is an association which connects a goal to a task and vice versa. It is an n-ary association

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public Task2GoalRelation	Public EMODENamedElement	
Association Task2GoalRelationGoal Bi-Directional	Public task2GoalGoal Goal	Public task2GoalGoalAssociation Task2GoalRelation	Connects a Goal to its Association
Association Task2GoalRelationTask Bi-Directional	Public task2GoalTaskAsoctiation Task2GoalRelation	Public task2GoalTask TaskNode	Connects the task to a goal
NoteLink Source -> Destination	Public Note	Public Task2GoalRelation	

TaskControlEdge

Type: **Class TaskEdge, UMLControlFlow**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Task *Keywords:*
Detail: *Created on* 13.06.2006. *Last modified on*.13.06.2006.
GUID: {1EDE5871-B97A-4589-A239-93978074AE88}

This task edge transfers the control flow

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public TaskControlEdge	Public UMLControlFlow	
Generalization Source -> Destination	Public TaskControlEdge	Public TaskEdge	

TaskEdge

Type: **Class PatternParticipationElement, UMLActivityEdge**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Task *Keywords:*
Detail: *Created on* 07.03.2006. *Last modified on*.13.06.2006.
GUID: {77E01A0E-4FAD-48f9-9B6A-941C63CD4B97}

A TaskEdge is a special connector connecting tasks in EMode. Its guard condition may have ModalityRestrictions attached.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public TaskEdge	Public PatternParticipation Element	
Generalization Source -> Destination	Public TaskControlEdge	Public TaskEdge	
Generalization Source -> Destination	Public TaskObjectEdge	Public TaskEdge	
Generalization Source -> Destination	Public TaskEdge	Public UMLActivityEdge	

Attributes

Attribute	Notes	Constraints and tags
InteractionFlags EMODEInteractionFlag Public [0..*]	Unary relations defined by the user can be attached to TaskEdges.	*Default:*
NecessaryEdge boolean Public TRUE	Whether the edge is necessary to start the next activity. If directed into and activity or merge/decision node, the user has to take care in the implementation that this works. Going into a join node, an non-active edge is taken to "be false".	*Default:* TRUE

TaskNode

Type: **Class PatternParticipationElement, UMLStructuredActivityNode**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Task *Keywords:*
Detail: *Created on* 07.03.2006. *Last modified on*.21.04.2006.
GUID: {8EF9EC39-78E0-461a-A850-404680C3523A}

This is a specialization for task diagrams. A structured task node is a container for other tasks.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association TaskIsExecutedBy Bi-Directional	Public theTaskNode2BeExec TaskNode	Public theFCACallToExecATaskNode FCACall	The connection between a system task and a FCA call
NoteLink Source -> Destination	Public Note	Public TaskNode	
NoteLink Source -> Destination	Public Note	Public TaskNode	
Generalization Source -> Destination	Public TaskNode	Public PatternParticipationElement	
Association Task2GoalRelationTask Bi-Directional	Public task2GoalTaskAssoctiation Task2GoalRelation	Public task2GoalTask TaskNode	Connects the task to a goal
Association TaskRelationSource Bi-Directional	Public theSourceRelationEnd TaskRelation	Public theSourceTasks4Relation TaskNode	Relates a TaskRelation to the source tasks
Association TaskRelationTarget Bi-Directional	Public theTargetRelationEnd TaskRelation	Public theTargetTasks4Relation TaskNode	The target tasks of a relation
Generalization Source -> Destination	Public TaskNode	Public UMLStructuredActivityNode	
NoteLink Source -> Destination	Public Note	Public TaskNode	
Generalization	Public	Public	

Connector	Source	Target	Notes
Source -> Destination	TaskNodeReference	TaskNode	
Association ReferencedTaskNode Bi-Directional	Public theReferenceToTheTaskNode TaskNodeReference	Public theReferencedTaskNode TaskNode	A reference to a TaskNode in order to provide the same functionality
NoteLink Source -> Destination	Public Note	Public TaskNode	
Association TaskIsEnactedBy Bi-Directional	Public theTaskNode2BeEnacted TaskNode	Public theAUIInteractor2EnactTheTask AUIInteractor	Describes the connection between task nodes and AUI components, which the interaction layout. This is a tight relationship, since the one (tasks) describe timely behaviour, whereas the other describe the layout.
Association TaskIsInteractedAt Bi-Directional	Public theTaskToBeInteracted TaskNode	Public theSpace4Interaction AUISpace	Details, where the interaction should take place

Attributes

Attribute	Notes	Constraints and tags
Description string Public	A text describing the task for the user	*Default:*
FastExit boolean Public FALSE	All currently running subtasks are terminated immediately after a token has reached a final node (true). If false, the current subtask (including ALL its children if it is a structured node) operation has to be finished and their tokens followed to the next stops (e.g. to have more than one output of a task).	*Default:* FALSE

Attribute	Notes	Constraints and tags
Flags EMODEInteractionFlag Public [0..*]	These flags are unary operators identified by UJF	*Default:* [isStatic = false]
PostCondition string Public [0..1]	A post condition for this element	*Default:*
PreCondition String Public [0..1]	Precondition for executing this element	*Default:*
Priority int Public 5	Presents the priority of the corresponding task. It is a symbolic measure for the level of importance from 0 (almost not important) to 10 (very important).	*Default:* 5

TaskNodeReference

Type: **Class TaskNode**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Task *Keywords:*
Detail: *Created on* 21.06.2006. *Last modified on*.21.06.2006.
GUID: {C25408DE-D7C4-49fc-BBE3-66E04AD57940}

References a different TaskNode. When this TaskNode is executed, the elements contained in the referenced node are invoked, but the in- and outcoming edges are used from this node.

E.g., multiple instances of "save item".

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public TaskNodeReference	Public TaskNode	
NoteLink Source -> Destination	Public Note	Public TaskNodeReference	
NoteLink Source -> Destination	Public Note	Public TaskNodeReference	
Association ReferencedTaskNode Bi-Directional	Public theReferenceToTheTaskNode TaskNodeReference	Public theReferencedTaskNode TaskNode	A reference to a TaskNode in order to provide the same functionality

Attributes

Attribute	Notes	Constraints and tags
ownedActivityEdges UMLActivityEdge Public [0..*]	The TaskEdges contained in the referenced TaskNode	*Default:*
ownedActivityNodes UMLActivityNode Public [0..*]	The TaskNodes contained in the referenced TaskNode	*Default:*

TaskObjectEdge

Type: **Class TaskEdge, UMLObjectFlow**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Task *Keywords:*
Detail: *Created on* 13.06.2006. *Last modified on*.13.06.2006.

GUID: {633C8366-3291-4aa4-9CA4-171556EF9F48}

This TaskEdge transfers object flows

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public TaskObjectEdge	Public UMLObjectFlow	
Generalization Source -> Destination	Public TaskObjectEdge	Public TaskEdge	

TaskPartition

Type: **Class EMODENamedElement, UMLActivityPartition**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Task *Keywords:*
Detail: *Created on* 24.03.2006. *Last modified on*.13.06.2006.
GUID: {F8B867D0-CD85-40f5-B075-CD00D3360A6D}

This is a partition upon which one may place edges, goals and tasks.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Generalization Source -> Destination	Public TaskPartition	Public UMLActivityPartition	

Connector	Source	Target	Notes
Association GoalsInPartition Bi-Directional	Public theGoalOfThePartition Goal	Public thePartitionOfTheGoal TaskPartition	
NoteLink Source -> Destination	Public Note	Public TaskPartition	
Generalization Source -> Destination	Public TaskPartition	Public EMODENamedElement	

Attributes

Attribute	Notes	Constraints and tags
type TaskPartitionType Public	Represents the type of the partition.	*Default:* [isStatic = false]

TaskPartitionType

Type: **Enumeration**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Task *Keywords:*
Detail: *Created on* 07.03.2006. *Last modified on*.13.06.2006.
GUID: {3D038F81-32EB-4523-A732-716C09A198D1}

Tasks and TaskGoals may be devided into 3 categories: system, ui and user.

Custom Properties

- isActive =

Attributes

Attribute	Notes	Constraints and tags

Attribute	Notes	Constraints and tags
System Public		*Default:*
Interactive Public		*Default:*
User Public		*Default:*

TaskRelation

Type: **Class EMODENamedElement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Task *Keywords:*
Detail: *Created on* 14.06.2006. *Last modified on*.28.06.2006.
GUID: {82483CC9-ACD9-4b9b-86A9-2EB870802C9E}

A relation between tasks. A group of source tasks can be related to a group of target tasks.

Custom Properties

- isActive =

Tagged Values

- isAbstract = false.

Connections

Connector	Source	Target	Notes
Association TaskRelationSource Bi-Directional	Public theSourceRelationEnd TaskRelation	Public theSourceTasks4Relation TaskNode	Relates a TaskRelation to the source tasks

Connector	Source	Target	Notes
Generalization Source -> Destination	Public TaskRelation	Public EMODENamedElement	
Association TaskRelationTarget Bi-Directional	Public theTargetRelationEnd TaskRelation	Public theTargetTasks4Relation TaskNode	The target tasks of a relation

Attributes

Attribute	Notes	Constraints and tags
SourceRole string Public	The role name for the source tasks	*Default:*
TargetRole string Public	The role name of the target tasks	*Default:*

