WORKPACKAGE 1: META-MODELLING

D1.3

USIXML DEFINITION

	Project acronym: UsiXML Project full title: User interface eXtensible Mark-up Language ITEA label n° 08026

WP Leader / Task Leader	THALES INTERNAL DOCUMENT NUMBER	PAGE
UCL / UCL	61 566 104/179/13	1/121
	Revision	**1**

***Deliverable N°**: D1.3*
***Due Date**: 11/2010*
***Delivery Date**: 05/2012*

***Short Description**: UsiXML definition (requirements, semantics, syntaxes, and stylistics) with UML class diagrams and XML schemas (initial version and two annual updates)*

***Lead Partner**: UCL*
***Contributors:** PRO**,** TID, UCL, UCLM, UJF, UPV, UND*
***Made available to**: Restricted*

Rev	**Date**	**Author**	**Checked by**	**Internal Approval**	**Description**
0.1	02/02/10	François Beuvens, Jean Vanderdonckt (UCL)	Jean Vander-donckt (UCL)	Jean Vander-donckt (UCL)	Initial version: table of contents
0.2	02/10/10	Jose Cantera, Jose Luis (TID) François Beuvens, Jean Vanderdonckt (UCL)	Jean Vander-donckt (UCL)	Jean Vander-donckt (UCL)	Incorporation of initial versions of AUI, CUI, and domain models
0.3	06/30/10	Contributions integrated from: TID, UPV	Jean Vander-donckt (UCL)	Jean Vander-donckt (UCL)	Several edits made
0.4	30/09/10	Ricardo Tesoriero (UCL & UCLM)	Jean Vander-donckt (UCL)	Jean Vander-donckt (UCL)	Incorporation of new task and context meta-models
0.5	28/10/10	Nathalie Aquino (UPV), Jean Vanderdonckt (UCL)	Jean Vander-donckt (UCL)	Jean Vander-donckt (UCL)	- Update of transformation meta-model - Submitted to audio conference of 28th Oct 2010
0.5.1	30/11/10	Nathalie Aquino (UPV), Ignacio Panach (UPV), Víctor López Jaquero (UCLM)			- Incorporation of comments from UCLM, UJF, UCL, ICI, ITB, UND, and UPV after audio conference of 28th Oct 2010 - Incorporation of core transformation rules definition - Connection with ATL and graph transformations
0.5.2	12/08/10	Mohamed Boukhebouze (UND), Philipe Thiran (UND)			Incorporation of Domain meta-model, reviewed after the Paris' meeting (10/06/2010)
0.6	30/01/11	François Beuvens (UCL), Jérémie Melchior (UCL), Jean Vanderdonckt (UCL), Ricardo Tesoriero (UCLM) Mohammed Boukhebouze (UND), Ignacio Panach (UPV)			Update of meta-model after general assembly (Namur 14 – 17 déc. 2010)

1.0	02/02/11	François Beuvens (UCL), Jérémie Melchior (UCL), Jean Vanderdonckt (UCL), Ricardo Tesoriero (UCLM), Philippe Thiran (UND), Mohamed Boukhebouze (UND)			Taking into account the latest discussion on Task MM after the audioconference with Ricardo Tesoriero and UCL. Update of the Abstract, Concrete, Domain, and Worklfow MM.
	11/04/2011	Alfonsa Garcia Frey (UJF)			Quality meta-model
	06/06/2011	Jean Vanderdonckt, François Beuvens, Jérémie Melchior, Ricardo Tesoriero, Vi Tran			Update of Abstract UI after meetings with Defimedia (06/06/2011, 30/05/2011, 23/05/2011, 16/05/2011)
	04/09/2011	François Beuvens, Jérémie Melchior, Ricardo Tesoriero, Alfonso G. Frey			Update of Task, Context, AbstractUI, and Quality meta-model Addition of a first version of the Mapping meta-model
1.2	09/11/2011	François Beuvens, Jérémie Melchior			Update Task, Context, Domain, Workflow, Abstract UI and Concrete UI meta-model.
1.3	22/11/2011	François Beuvens, Jérémie Melchior			Update and document the Final version of Task and Abstract UI meta-models
1.4	15/03/2011	François Beuvens, Jérémie Melchior, Jean Vanderdonckt	D. Faure, THA		Update of graphical part of CUI
1.4.1	22/05/2011	François Beuvens	D. Faure, THA		Minor updates
1.4.3	06/02/2013	Marc Gil (PRO), Javier Cano (PRO)			Appendix for modifications on AUI metamodel to make it EMF-oriented

CONTENTS

Table of Contents

1. INTRODUCTION

This document is aimed at describing and defining the models and meta-models of UsiXML.

2. METHODOLOGICAL FRAMEWORK

2.1. Global rationale for meta-modelling

2.1.1. Cameleon Reference Framework

The CAMELEON Unified Reference Framework was produced by the EU-funded CAMELEON FP5 Project (FP5-IST4-2000-30104). CAMELEON [CCB02] [CCTLBV03], describes a framework that serves as a reference for classifying user interfaces supporting multiple targets, or multiple contexts of use in the field of context-aware computing. The CAMELEON Framework provides a unified understanding of context-sensitive user interfaces rather than a prescription of various ways or methods of tackling different steps of development. Rather, the framework structures the development life cycle into four levels of abstraction, from the task specification to the running interface (see Figure 1):

- The **Task and Concepts level** (corresponding to the Computational-Independent Model–CIM–in MDE) which considers the logical activities that need to be performed in order to reach the users' goals. Often they are represented hierarchically along with indications of the temporal relations among them and their associated attributes.
- The **Abstract User Interface** (AUI) (corresponding to the Platform-Independent Model–PIM– in MDE) is an expression of the UI in terms of interaction spaces (or presentation units), independently of which interactors are available and even independently of the modality of interaction (graphical, vocal, haptic …). An interaction space is a grouping unit that supports the execution of a set of logically connected tasks.
- The **Concrete User Interface** (CUI) (corresponding to the Platform-Specific Model–PSM– in MDE) is an expression of the UI in terms of "concrete interactors", that depend on the type of platform and media available and has a number of attributes that define more concretely how it should be perceived by the user. "Concrete interactors" are, in fact, an abstraction of actual UI components generally included in toolkits.
- The **Final User Interface** (FUI) (corresponding to the code level in MDE) consists of source code, in any programming language or mark-up language (e.g. Java or HTML5). It can then be interpreted or compiled. A given piece of code will not always be rendered on the same manner depending on the software environment (virtual machine, browser …). For this reason, we consider two sublevels of the FUI: the source code and the running interface

These levels are structured with a relationship of reification going from an abstract level to a concrete one and a relationship of abstraction going from a concrete level to an abstract one. There can be also a relationship of translation between models at the same level of abstraction, but conceived for different contexts of use. These relationships are depicted on the figure below.

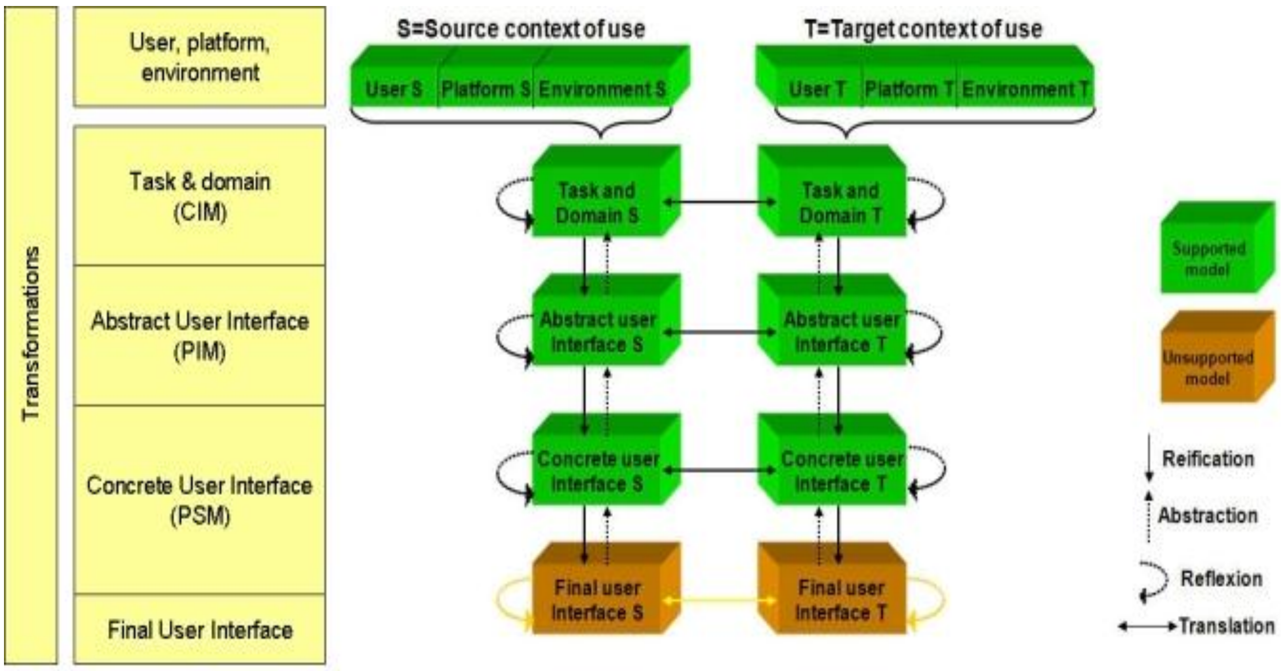


Figure 1 - Relationships between components in the Cameleon Reference Framework

2.1.2. Multipath development of user interfaces

Concerning the level of abstraction where a user has to begin the sketching of its interface, we don't want to impose anything. The user should have the choice to start from any point: Task & domain level, Abstract UI level or Concrete UI level.

Beginning and thinking with the most abstract level (tasks) seems to be the best way to design the best interfaces. However, the wish of the user could be to start at another level of abstraction, because of other needs or constraints like time. In this case, we want to provide to the user the possibility of choosing.

2.1.3. Self-containment of user interfaces meta-models

This idea directly follows the previous point. In order to provide the possibility of choosing the starting level of abstraction, each meta-model must be independent of each other. At any level, all the information needed to derive a Final User Interface should be retrieved.

Nevertheless, each model can contain less or more details in function of their level of abstraction. For example, the concept of behaviour defined at Abstract User Interface level is also defined for the Concrete User Interface and can be specified at this level by taking into account a graphical interaction for instance.

2.2. Methodological principles for meta-modelling

Several principles have to be followed in order to define high-quality meta-models. We hereby define meta-model element by referring to any entity, attribute or relationship of a meta-model. Each principle will be consistently described according to the following template:

- **P**[id]**: [Principle name].**
  General definition: General definition of the principle according to computer science.

<u>*Specific definition*</u>: Specific definition of the principle in human-computer interaction regarding UsiXML language. The principle should be stated in terms of ability.
<u>*Positive Example*</u>: Appropriate application of the principle (synonym: do's).
<u>*Negative Example*</u>: Inappropriate application of the principle (synonym: don't).
<u>*Exception*</u>: Situation in which the principle may not be applied.
<u>*Synonym*</u>: Any other name for the same principle.

These ones are summarized in the followings:

- **P1: Completeness.**
  <u>*General definition*</u>: In a general way, we can say that an object is complete if nothing needs to be added to it.
  <u>*Specific definition*</u>: Regarding UsiXML, completeness means the ability of a meta-model to abstract all features of the domain via appropriate concepts and relations.
  <u>*Positive Example*</u>: If we focus on the graphical refinement of the Concrete User Interface model, all the possible components have to be described in order to allow the derived model to describe any user interface.
  <u>*Negative Example*</u>: Let's assume a meta-model holding an entity *car* with no attribute or component allowing describing its color. Then the meta-model will not be complete because if a model has to describe the color of the car it will not be possible.
  <u>*Exception*</u>:
  <u>*Synonym*</u>:
- **P2: Consistency.**
  <u>*General definition*</u>: Agreement or accordance with facts, form, or characteristics previously shown or stated.
  <u>*Specific definition:*</u> Regarding UsiXML, consistency means the ability of a meta-model to produce an abstraction in a way that reproduce the behaviour of the features of the domain in the same way throughout the meta-model and that preserves this behaviour throughout any manipulation of the meta-model.
  <u>*Positive Example*</u>: In the Concrete User Interface meta-model, we can define a behaviour for a component. The way the behaviour is associated to a component is the same for another component. Furthermore, if a component is modified (for example, by changing its name), its behaviour is not impacted.
  <u>*Negative Example*</u>: Let's assume a meta-model in which an entity *car* is linked with an entity *race*. The car entity has an attribute *numberRaces*. Inconsistency can occur because the number of races can be retrieved through the relationship between the two entities and the *numberRaces* attribute adds unnecessary information that can be false.
  <u>*Exception*</u>:
  <u>*Synonym*</u>:
- **P3: Correction.**
  <u>*General definition*</u>: State in which no mistake exists.
  <u>*Specific definition*</u>: Regarding UsiXML, correction means the ability of a meta-model to produce an abstraction in a way that correctly reproduces the behaviour of the features of the domain.
  <u>*Positive Example*</u>: In the Concrete User Interface model, if a component « checkbox » is defined, it must have all the features commonly present in the usual representation of the checkboxes (i.e. Little boxes that you can check).
  <u>*Negative Example*</u>: Let's assume a meta-model in which an entity *car* is defined with 10 wheels, then it does not match the reality.

Exception: Sometimes for any reason we do not want to model the real world.
Synonym: Soundness

- **P4: Separation of concerns.**
  General definition: In computer science, separation of concerns is the process of separating a computer program into distinct features that overlap in functionality as little as possible.
  Specific definition: Regarding UsiXML, separation of concerns means the ability of the meta-models to address only one specific point without overlapping with the other meta-models. The separation of concerns is also important inside the meta-models, where the different concepts must be well separated.
  Positive Example: Task concepts are present only on the Task meta-model level and can not appear in other meta-models.
  Negative Example: Let's assume that the modality would taken into account in the Abstract User Interface meta-model, then it would overlap with the Concrete User Interface meta-model and avoid good separation of concerns.
  Exception:
  Synonym:
- **P5: Packaging.**
  General definition: A package is a way of grouping entities together, which can be useful for understanding and managing a model.
  Specific definition: Regarding UsiXML, packaging means the ability of each meta-model to be composed from several semantically related parts called packages.
  Positive Example: In the Concrete User Interface meta-model, we have one package for each interaction modality.
  Negative Example: Let's assume a meta-model describing cars and races. Defining cars as a characteristic of races could work but would not be well separated for understanding and managing. A better way to represent should be to define cars and races separately and link them together through relationship.
  Exception:
  Synonym: Clustering
- **P6: Extensibility.**
  General definition: In software engineering, extensibility is a system design principle where the design takes into consideration future growth.
  Specific definition: Regarding UsiXML, extensibility means the ability of each meta-model to be easily extended with new meta-model element. Further, it concerns also the whole UsiXML language for which new meta-model can be easily added in order to address specific need.
  Positive Example: Add a new modality in the Concrete User Interface is easily done by refining the *CuiContainer*, the *CuiInteractor* and the *CuiRelationship*.
  Negative Example: Let's assume a meta-model describing cars and races, with these two concepts modeled on an entity *car* and an entity *races*, and linked through some relationship. To model a new engine would be really convenient because a new entity would be created (for example *moto*) with all the features needed. A better way to model that would be to create an entity *engine* with possibilities of refinement and for which all the features inherent to an engine would be already present.
  Exception:
  Synonym:
- **P8: Pragmatism.**
  General definition: Pragmatism is the way of dealing with a problem in a realistic way rather than obeying fixed theories, ideas or rules.

Specific definition: Designing the meta-models of UsiXML has to take into account the developers and their necessities.
Positive Example: In the Concrete User Interface model, several graphical components are defined in order to provide a maximum powerfulness to the user. All these component are feasible.
Negative Example: Some components are maybe not feasible in a generic way or would take too many resources. For example a component able to predict the future would be very convenient but totally impossible for the moment.
Exception:
Synonym:

- **P11: Pattern based meta-modelling.**
  General definition: Design pattern is a description or template for how to solve a problem that can be used in many different situations. Patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.
  Specific definition: Regarding UsiXML, pattern based meta-modelling means the ability for each meta-model to respect the following patterns: (i) The name of the root entity represents the name of the model; (ii) Fields of values are preferably represented explicitly through enumerations.
  Positive Example: In the Concrete User Interface meta-model, the root entity is *CUIModel* or the different possible *actionType* (attribute of the entity *Action*) are listed in an enumeration *ActionType*.
  Negative Example: Typing the attribute *actionType* as a string would let it free, without constraint.
  Exception:
  Synonym:
- **P15: Expressiveness.**
  General definition: Quality of being expressive, of being able to express any possibility.
  Specific definition: Regarding UsiXML, expressiveness means the ability of a meta-model to express via an abstraction any feature of the domain.
  Positive Example: In the Concrete User Interface meta-model, the *CuiInteractor* can abstract any interactor for a user interface.
  Negative Example:
  Exception:
  Synonym:

2.3. Methodological principles for modelling

Several principles have to be followed in order to define high-quality models. We hereby define model element by referring to any entity, attribute or relationship of a model. Each principle will be consistently described according to the following template:

- **P**[id]**: [Principle name].**
  General definition: General definition of the principle according to computer science.
  Specific definition: Specific definition of the principle in human-computer interaction regarding UsiXML language. The principle should be stated in terms of ability.
  Positive Example: Appropriate application of the principle (synonym: do's).
  Negative Example: Inappropriate application of the principle (synonym: don't).
  Exception: Situation in which the principle may not be applied.
  Synonym: Any other name for the same principle.

These ones are summarized in the followings:

- **P12: Completeness. ??**
  General definition: In a general way, we can say that an object is complete if nothing needs to be added to it.
  Specific definition: Ability of a model to fulfill all the requirements needed to describe its features.
  Positive Example: If we focus on the graphical refinement of the Concrete User Interface model, it has to address all the possibilities for designing a graphical interface.
  Negative Example: Let's assume an attribute *color* for a component representing a car. If only the red is possible, then all the choices are not represented and the model is not complete.
  Exception:
  Synonym:
- **P13: Consistency. ??**
  General definition: Agreement or accordance with facts, form, or characteristics previously shown or stated.
  Specific definition: Regarding UsiXML, consistency means the ability of a model to produce an abstraction in a way that reproduce the behaviour of the features of the domain in the same way throughout the model and that preserves this behaviour throughout any manipulation of the model.
  Positive Example:
  Negative Example:
  Exception:
  Synonym:
- **P14: Correction. ??**
  General definition: State in which no mistake exists.
  Specific definition: Regarding UsiXML, correction means the ability of a model to produce an abstraction in a way that correctly reproduces the behaviour of the features of the domain.
  Positive Example:
  Negative Example:
  Exception:
  Synonym: Soundness
- **P6: Extensibility.**
  General definition: In software engineering, extensibility is a system design principle where the design takes into consideration future growth.
  Specific definition: Regarding UsiXML, extensibility means the ability of each model to be easily extended with new model element.
  Positive Example: In the Concrete User Interface model, an interface can easily be extended by composing it with other interface (via the *AuiContainer*).
  Negative Example:
  Exception:
  Synonym:
- **P15: Expressiveness.**
  General definition: Quality of being expressive, of being able to express any possibility.
  Specific definition: Regarding UsiXML, expressiveness means the ability of a model to express via an abstraction any feature of the domain.
  Positive Example: In the Concrete User Interface model, it is possible to add an entity to abstract any modality.
  Negative Example:
  Exception:
  Synonym:

- **P16: Concision.**
  General definition: State or quality of being concise, to only provide needed information.
  Specific definition: Regarding UsiXML, concision means the ability of a model to produce concise, compact abstractions to abstract real world aspects of interest.
  Positive Example: In the Concrete User Interface model, enumerations are used in order to only propose possible choices.
  Negative Example:
  Exception:
  Synonym:
- **P17: Separability.**
  General definition: The capability of being separated.
  Specific definition: Regarding UsiXML, separability means the ability of models to univocally classify any abstraction of a real world aspect of interest into one single model.
  Positive Example: Task model and Workfow model are both at the same level abstraction but they capture different information.
  Negative Example:
  Exception:
  Synonym:
- **P18: Correlability.**
  General definition: The capability of being correlated, of being linked.
  Specific definition: Regarding UsiXML, correlability means the ability of models to univocally and unambiguously establish relationships between models to represent a real world aspect of interest.
  Positive Example: The Mapping model is used to link two models from two different layer of abstraction.
  Negative Example:
  Exception:
  Synonym:
- **P19: Integrability.**
  General definition: The quality of being integrated.
  Specific definition: Regarding UsiXML, integrability means the ability of models to concentrate and integrate abstractions of real world aspects of interest into a single model or a small list of them.
  Positive Example: In the Concrete User Interface model, it is easy to add new modalities to an interface, or to add new features to existing modalities.
  Negative Example:
  Exception:
  Synonym:
- **P20: Usefulness.**
  General definition: The quality of being of practical use.
  Specific definition: Regarding UsiXML, usefulness means the ability of the models to avoid describing elements that does not convey any information to any problem feature.
  Positive Example: In the Concrete User Interface model, the graphical refinement only describes existing and useful components.
  Negative Example:
  Exception:
  Synonym:
- **P21: Pragmatism**
  General definition: Pragmatism is the way of dealing with a problem in a realistic way rather than

obeying fixed theories, ideas or rules.
Specific definition: Designing the models of UsiXML has to take into account the developers and their necessities.
Positive Example: In the Concrete User Interface, describe a component that would be able to render the smell of the food.
Negative Example:
Exception:
Synonym:

3. META-MODELS

3.1. User Interface

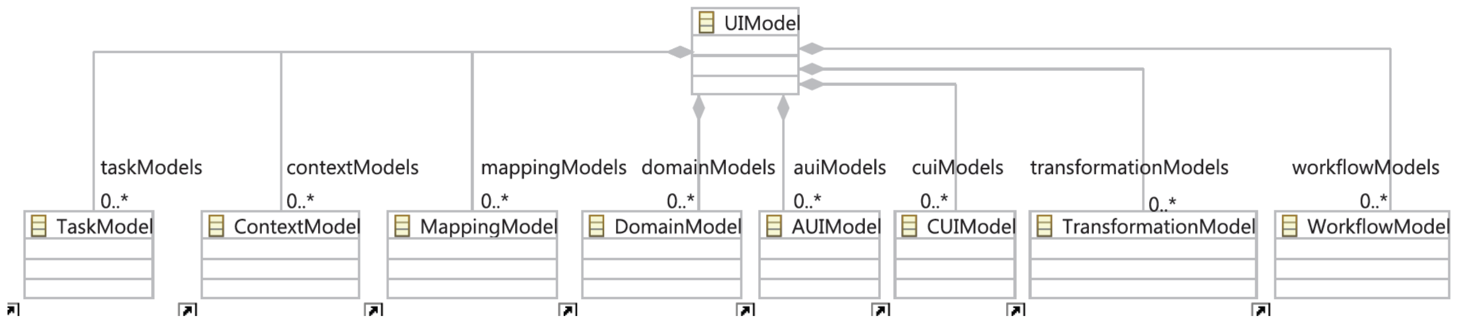


Figure 2. UIModel Overview.

The UI meta-model is the UI model composed by all the models, as in Figure 2. There can be any of the models and they can be combined together. Each model can have several instances.
The following models will be described in details in this document: TaskModel, ContextModel, DomainModel, TransformationModel, AUIModel, CUIModel and WorkflowModel.
The MappingModel will be introduced in the document but not into details.

3.2. Task

3.2.1. Overview

The task metamodel defines the abstract syntax used to represent the task model of the system to be developed. It is part of the Tasks & Concepts layer of the UsiXML Framework. From the structural perspective, it is inspired by the HTA approach where complex tasks are decomposed into subtasks; from the behavioral perspective, it is inspired on CTT where temporal relationships are defined in terms of LOTOS operators.

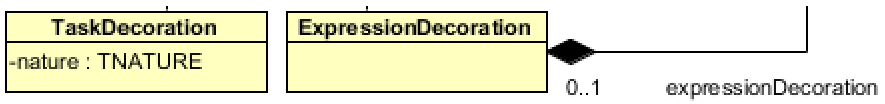


Figure 3 shows an overview of the task metamodel.

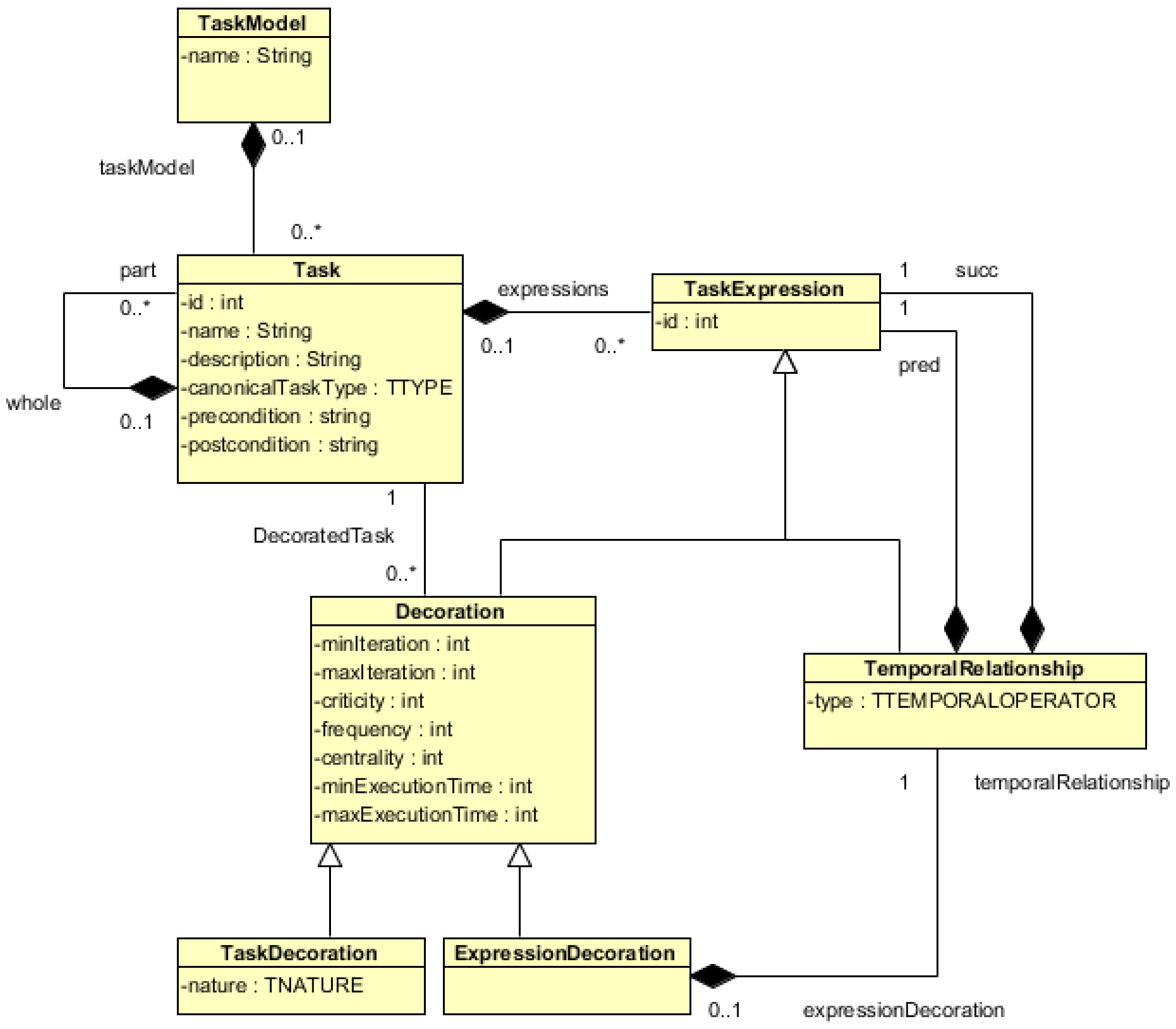


Figure 3: Task Model overview

3.2.2. Main entities

- *TaskModel:* The TaskModel meta-class describes the interaction between the entities that are parts of the system, and the system itself, in terms of tasks that are performed by the entities of the system. It defines a set of tasks as a whole that are decomposed into sub-tasks as parts structuring the system into task hierarchies that define the behavior of the system.
  - *Attributes:*
    - name **(String)**: Defines the name of the task model.
      - Example:
      - Location-aware remote control, Touristic Guide.
  - *Example:*
    - Let Location-aware remote control be a TaskModel of a remote control application that mutates the control UI layout according to the nearest device to control.
- *Task:* The Task meta-class describes a task performed by one or more system entities. From the structural point of view, a task can be atomic or composed. While composed tasks represent complex tasks that can be decomposed into simpler sub-tasks; atomic tasks represent an indivisible action that is performed by a single

entity of the system. Thus, collaborative tasks are represented as composed tasks that can be decomposed into atomic tasks performed by single entities. To define the temporal relationship among subtasks, the parent task defines a set of expressions that describes the temporal relationships among subtasks. The tasks can also be decorated in order to give information such as their criticity, optionality and other details.

- *Attributes:*

  - id **(int)**: Defines the task unique id.

  - name **(String)**: Defines the task name.

    Example:

  - NotifyPointOfInterest, NextPhoto, PreviousVideo, EnterComment, etc.

- - description **(String)**: Explains what the task is about by a textual description as complete as possible.

    Example:

  - NotifyPointOfInterest shows a notification to the user that a point of interest has been reached.

- - canonicalTaskType **(TASKTYPE)**: The canonical type of a task describes the action of the task without going deeper on details on how it applies.

  - Possible value:
  - CONVEY, CREATE, REINITIALIZE, FILTER, DELETE, DUPLICATE, NAVIGATE, PERCEIVE, MOVE, MODIFY, MEDIATE, SELECT, TRIGGER, STOP, TOGGLE
  - Example:
  - The SignGuestbook task has as the attribute set to APPEND. It adds a sign to a guest book.

- - precondition **(String)**: Describe the condition in which the task can be executed.

    Example:

  - NotifyPointOfInterest: a demand for a point of interest must have been created.

- – postcondition **(String)**: Describe how the task changes the system.

    Example:

  - NotifyPointOfInterest: a message is shown to the user with the notification.

- *Example:*

  - Let SignGuestbook be a Task where the id is 1, with the name SignGuestbook, the description is to sign the guest book.

➢ *TaskExpression:* The TaskExpression is an abstract meta-class that defines expressions on tasks. Task expressions allow developers to define: (a) task attributes that varies according to the situation (i.e. criticity, frequency, etc.), and (b) temporal relationships among tasks. To represent a TaskExpression we employ the Composite design pattern where the TaskExpression plays the role of Component.

➢ *TemporalRelationship:* The TemporalRelationship meta-class is a TaskExpression that allows developers to define the temporal relationship among TaskExpressions. TemporalRelationships are temporal operations defined as LOTOS operators on TaskExpressions. The LOTOS temporal operation is defined as a TTEMPORALOPERATOR that defines the ENABLING, CONCURRENCY, DISABLING, SUSPEND, ORDERINDEPENDENCE and CHOICE temporal operations. The TemporalRelationship meta-class plays the role of Composite in the Composite design pattern rooted on TaskExpression.

- *Attributes:*

  - type **(TTEMPORALOPERATOR)**: Defines the type of the temporal operator in a TemporalRelationship.

  - Possible values:
  - ENABLING, CONCURRENCY, DISABLING, SUSPEND, ORDERINDEPENDENCE and CHOICE

- Example:
- Let suppose that EnterInformation and SubmitInformation are two tasks that are decorated, and we want to express that the EnterInformation task is performed before SubmitInformation task. Then, the TemporalRelationship defines EnterInformation task decoration as predecessor and the SubmitInformation task decoration as successor. The TTEMPORALOPERATOR used is ENABLING.

➢ *Decoration:* The Decoration is an abstract meta-class of TaskExpression that allows developers to define TaskExpression attributes that vary according to the situation they are performed (i.e. criticity, frequency, etc.). The Decorator plays the role of abstract Component in the Composite design pattern rooted on the TaskExpression meta-class. It has two concrete meta-classes according to the type of element they are decorating (Task or TaskExpression).

o Attributes:

- minIteration **(Integer)**: Defines the minimum cardinality of the iteration for the task or task expression. Note that if this attribute is set to 0 then the task or task expression it refers to is optional.

- Example: If this attribute is set to 3, then the task, or the expression, it refers to must be repeated at least 3 times

- maxIteration **(Integer)**: Defines the maximum cardinality of the iteration for the task or task expression. Note that if this attribute is set to -1 then it means that the maximum cardinality is set to infinite.

- Example: If this attribute is set to 5, then the task, or the expression, it refers to may be repeated at most 5 times.

- criticity **(Integer)**: Defines the criticity level of a the task (from 0 to 10). If a task is critic, the criticity level will be high.

- Possible value: An integer from 0 to 5
- Example: A very critical task will have a criticity level around 4 or 5.

- frequency **(Integer)**: Defines the frequency of the task (from 0 to 10). If a task happens often, the frequency will be high.

- Possible value: An integer from 0 to 5
- Example: A task that will often happen will have a high frequency (4 or 5).

- centrality **(Integer)**: Defines if the task is important for the current context. A log in task will not be central for visiting a website, but would become central when the user want to access to his account.

- Possible value: An integer from 0 to 5

- minExecutionTime **(Integer)**: Set the minimum time for the execution.

- Possible value: Any integer.
- Example: The value 5 means that the task should at least last 5 seconds.

- maxExecutionTime **(Integer)**: Defines if the task is important for the current context. A log in task will not be central for visiting a website, but would become central when the user want to access to his account.

- Possible value: An integer from 0 to 5
- Example: The value 10 means that the task could not last more than 10 seconds.

➢ *TaskDecoration:* The TaskDecoration is a concrete meta-class of Decoration that allows developers to define Decorations on Tasks.

o Attributes:

- nature **(TNATURE)**: Defines the interaction needed for the task. This attribute depends on the type of entity that is available to perform the task. If it is a single user, the nature will be USER; if it is the system, the nature will be SYSTEM; if both are needed, the task will be INTERACTIVE; if the task is unknown or too complex to describe the nature, the UNDEFINED value is set.

- Possible values: UNDEFINED, USER, SYSTEM, INTERACTIVE
- Example: A user filling a form is an INTERACTIVE task because the user is interacting with the system in order to fill the form. A user that is thinking about a solution in his mind is an USER task. The printing of a document is a SYSTEM task because it only involved the system.

➢ *ExpressionDecoration:* The ExpressionDecoration is a concrete meta-class of Decoration that allows developers to define Decorations on TemporalRelationships.

3.3. Context

3.3.1. Overview

The context meta-model defines the abstract syntax used to represent the context model of the system to be developed. This is a transversal model because it is related to models belonging to all layers of the UsiXML Framework.

The context model is defined in terms of: Observables and Situations. While Observables describe the entities that affect the system, Situations describe which states of the entities affect the system, taking into account the space-time relationship among them.

The Figure 4 exposes the overview of the context meta-model defined in the UsiXML framework.

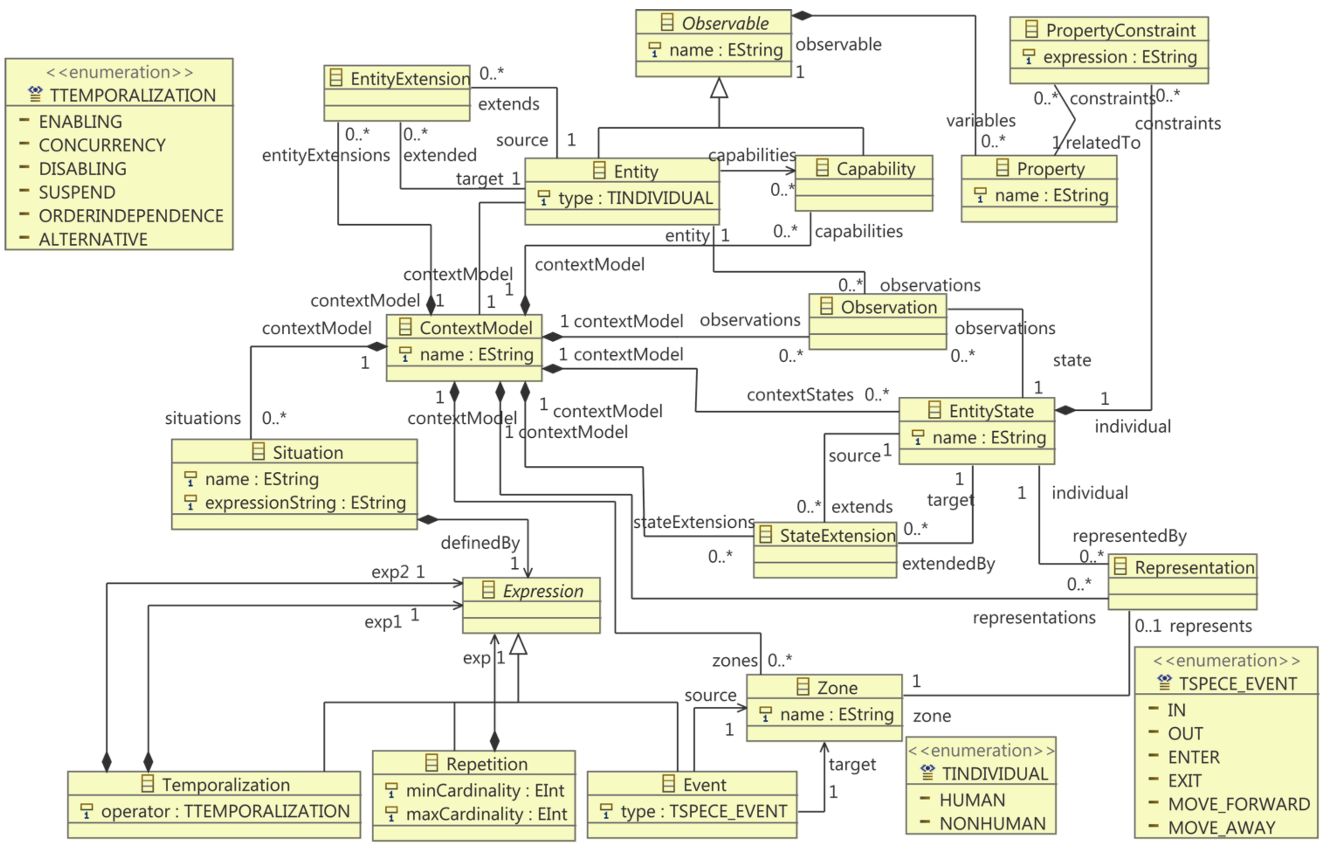


Figure 4 : Overview of the context model

3.3.2. Main entities

- ContextModel: The ContextModel meta-class defines the system context describing two aspects. On the one hand, it describes the Observable aspects of the environment that are relevant to the system operation. On the other hand, it describes the set of context Situations that are relevant to the system operation in terms of the state of the Observables.
  - *Attributes*:

    - name **(String)**: Location-aware remote control, Touristic Guide.
  - *Example*: Outdoor, Indoor, Cloud, etc.

- Observable: The Observable is an abstract meta-class that defines an aspect that is relevant to the system. It is defined in terms of quantifiable variables (ObservableVariables) that are used to hold the state of the Observables. There are two kinds of Observables: Entities and Capabilities. While Entities define those elements that affect the system operation (i.e. the mobile device being used to interact with the system), Capabilities describe the characteristics of the Entities (i.e. GPS be part of a car or a mobile phone). Every Observable is described in terms of Properties that hold the Observable state.
  - *Attributes:*

    - name **(EString)**: Identification for Entities and Capabilities
      - Example:
      - GPRSCapability, NFCCapability, DisplayCapability, DVDPlayer, etc.
      - Possible value: any non-empty string.

- *Property:* The Property meta-class represents a quantifiable attribute of an Observable that affects the system. A Property related to a set of ProperyConstrants that are part of a relevant state of an Entity defining a Situation.
    - *Attributes:*

      - name **(EString)**: Identification of the aspect

        - Example:
        - VerticalResolution, Speed, Range, Role, WorkExperience, Team, etc.
        - Possible value: any non-empty string.
    - *Example:*

  The Property usually describes an attribute of a capability, such the VerticalResolution or the HorizontalResolution of a DisplayCapability. Besides, Properties can be used to describe user skills or memberships, such as Role, Team or Experience. The Property concept provides a flexible way of modeling Capabilities such as those described in the Delivery Context Ontology. Jointly with the Observable sub-meta-classes, it is a powerful resource to improve the reusability of the models.

- *Entity:* The Entity meta-class defines an Observable of the system which state is attached to the situations it operates. There are two types of entities; those related to system aspects, and those related to human aspects. Entity characteristics are described in terms of Properties, or they can be grouped in Capabilities.
    - *Attributes:*

      - type **(TINDIVIDUAL)**: Defines the type of the entity.
    - Example: IPhone (NONHUMAN), Manager (HUMAN) etc.
    - Possible value: HUMAN or NONHUMAN.
    - *Example:*

      The Entity is usually employed to describe different types of devices, such as mobile phones, TVs, DVD players, etc. Besides, they can also be used to build different user profiles (User, Guest, Administrator, Professor, etc.), or any other metadata entity information that should be taken into account.

- *Capability:* The Capability meta-class defines a characteristic that can be contained in many Entities. The main advantage of grouping Properties in Capabilities is the possibility to build libraries using standard Capabilities, such as those defined by the Delivery Context Ontology. As result, the Context model can be reused in different applications.
    - *Example:*

      The Capability is usually employed as aspect descriptors, such as GPRSCapability, NFCCapability, WiredNetworkCapability, DisplayCapability, etc.

- *EntityExtension:* The EntityExtension meta-class represents a relationship between Entities where all Properties and Capabilities of the parent Entity are inherited by the Entity that extend it. It works in the same way as Class inheritance.

- *Observation:* The Observation meta-class represents a relationship between an Entity and an EntityState that is relevant to the system operation. Thus, the Properties defined by the Entity that is being observed, and the Properties defined by the Capabilities that are part of the Entity, can be constrained by the PropertyConstraints of the EntityState that belongs to the Observation.

➢ *EntityState:* The EntitySate meta-class represents a state of an entity of the environment that is relevant to the system operation. It is defined in terms of Observable Properties that are defined as PropertyConstraints. The EntityState, the EntityState is related to Zones through the Representation relationship in order to add space and time context to it.

- *Attributes:*

  - name **(EString)**: Identification for entity state

    - Example:
    - iPhoneOnWifi, iPhonePhotographer, ExperiencedPhotographer Manager, Engineer, TeamAPlayer, PrinterOnline, etc.
    - Possible value: any non-empty string.

PropertyConstraint: The PropertyConstraint meta-class describes the state of a Property belonging to an Observable aspect that is an Observation of an EntityState. Note that a PropertyConstraint is related to a single Property. However, the EntityState is defined by a set of PropertyConstraints, and the same Property can be constrained by more than one PropertyConstraint (i.e. the temperature Property may be constrained by the following PropertyConstraints: temperature >36 and temperature < 38). Besides, the PropertyConstraint meta-class is close related to the Observation relationship because PropertyConstaints must be defined between Properties belonging to Observables that are linked to the EntityState through an Observation relationship. In order to avoid aliases, the convention to name Properties in PropertyConstraints is: <name of Observable>.<name of Property>.

- *Attributes:*

  - expression **(EString)**: Represents the Property constraint. The name of the ObservableVariable to be quantified must be preceded by the name of the Observable it belong to and a point in order to avoid name conflicts.

- Example: Environment.temperature > 30°C, GPS.Enabled=true, Display.VResolution < 320 pixels, Connection.BitRate <10 Mbps, etc.
- Possible value: any non-empty string.

➢ *StateExtension:* The StateExtension meta-class defines a relationship between two EntityStates: the extended and the extension. The mechanism replicates the Observations and the PropertyConstraints from the extended EntityState to the extension EntityState. Thus, complex EntityStates can be built based on simple EntityStates representing the state of Entity aspects. As EntityStates can be extended by more than one EntityState, the EntityStates representing the states of the aspects can be reused to build different combinations of complex EntityStates. The Entities being Observations of EntityStates cannot be repeated to avoid conflicts.

➢ *Zone: The Zone element represents physical and virtual spaces in the environment. The Zone representation is able to define both, static and dynamic spaces. The meaning of the static space is related to spaces that do not change their position during the execution of the application, for instance, Home, Office, Grandplatz, Squares, and so on. Conversely, the dynamic spaces change their position during the execution of the application, for instance, the UserCloseZone, Car, Plane, Robot, etc. Spaces are usually related by events that eventually affect the system. Thus, Zones are close related to the SpaceEvent element because it establishes a relationship between two Zones. The Zone concept embraces both, virtual and physical zones. On the one hand, physical zones are physical representations of physical spaces (i.e. Car, Home, Office, Building, User, etc.). On the other hand, virtual zones are representations of virtual spaces (i.e. organization domain, process working space, Internet, Intranet, LAN,* *MAN, WAN, etc.). Note that an entity can be represented by more than one Zone. It is especially interesting when modeling "interaction zones". The*

- *Figure 5 shows an example of how the same entity is able to define different "interaction zones". Finally, the Anywhere Zone represents a kind of root Zone where all Zones are included.*
  - *Attributes:*

    - name **(EString)**: Identification of the interaction zone

    - Example:

      User close range, Advertisement panel medium range and building long range.
    - Possible value: any non-empty string.
  - *Example:*

The Zone is usually used to describe events, suppose that we have to know if a User is in a Room. The Room may be defined as a Zone (a static space in this particular case) and the User can be defined as an entity (a dynamic space). Thus, we can define the User Entity which does not have a particular state, for instance SingleUser state. Thus, through a Representation relationship, the SimpleUser is related to the CloseUserZone. The space event can be described as "CloseUserZone IN Room".

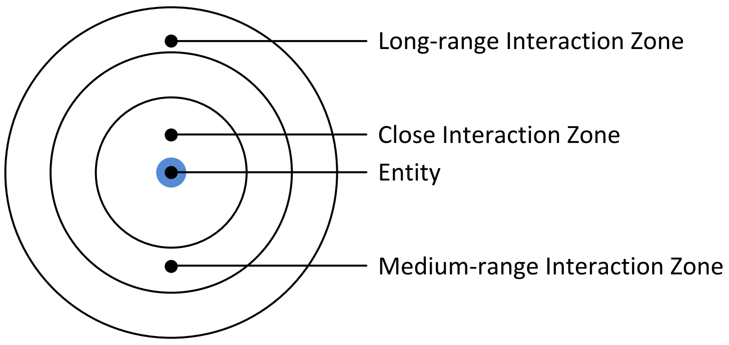


Figure 5: Interaction zone variations

- *Representation:* The Representation relationship relates EntityStates to Zones. It represents "interaction zones" that are used to define contextual situations. Although an EntityState can be represented by more than one Zone; a Zone represents at most one EntityState. The Representation is the only way to relate EntityStates to Situations by the means of SpaceEvents.

- *Situation:* The Situation meta-class identifies the space-time relationship among the different EntityStates of the entities that are part of the Entities that affect the system operation. Situations are the "interface" of the context to the rest of the models. The "mapping model" is in charge of linking other model elements to Situations in order to contextualize them. It represents a relevant context state for the system that is represented by an Expression.

  - *Attributes:*

    - name **(EString)**: Identification of the situation (space-time relationship among EntityStates)

    - Example:

      InFrontOfTheMonument, At Home, AsAdministrator.
    - Possible value: any non-empty string.

- *Expression:* The Expression meta-class represents relationships among EntityStates in space and time dimensions. While spatial relationships are represented by the Event sub-meta-class, the Temporalization Expression sub-meta-class defines the temporal relationships among spatial relationships.

- *Event*: The Event meta-class defines an Expression that represents the special relationships among the EntityStates that are relevant to the system operation. The relationship is established through the Zone meta-class that is related to the EntityState through the Representation relationship. By relating Zones, EntityStates represent the relationships among different Entities in the context. There are 6 types of Spatial relationships that are grouped into: trigger relationships (ENTER, EXIT), state relationships (IN, OUT), and directional relationships (MOVE_FORWARD, MOVE_AWAY).

  - *Attributes:*

    - type **(TSPACE_EVENT)**: Defines the type of the Event.

    - Example: User IN Shopping, Engineer OUT Office, Tourist MOVES_FORWARD Monument.
    - Possible values: IN, OUT, ENTER, EXIT, MOVE_FORWARD, MOVE_AWAY.

- *Repetition:* The Repetition meta-class represents the repetition or cardinality of the expression it embraces. It is defined by two integers that represent the minimum and maximum times the expression is repreated.

  - Attributes:

    - minCardinality **(Integer)**: Defines the minimum cardinality of the repetition for the Expression. Note that if this attribute is set to 0 then the embraced expression is optional. Besides, if this attribute is set to an integer greater than 0 then it is mandatory. Finally, no values below 0 are forbidden.

    - Example: If this attribute is set to 3, then the Expression it refers to must be repeated at least 3 times.
    - Possible values: Any integer greater or equals than 0

    - maxCardinality **(Integer)**: Defines the maximum cardinality of the repetition for Expression. Note that if this attribute is set to -1 then it means that the maximum cardinality is set to infinite.

    - Example: If this attribute is set to 3, then the Expression it refers to must be repeated at most 3 times.
    - Possible values: Any integer different than 0

- *Temporalization:* The Temporalization meta-class represents the temporal relationship among EntityStates that defines a Situation. The temporal relationships among Events are defined in terms of LOTOS temporal operators. Thus, by relating Events, we relate Zones; by relating Zones, we relate EntityStates, and by relating EntityStates we relate the entities that affect the system operation.

  - Attributes:

    - operator **(TTEMPORALIZATION)**: Defines the temporal relationship among Expressions.

    - Example: The ENABLING operator defined between *exp1* and *exp2*, means that *exp1* occurred before *exp2*.
    - Possible values: ENABLING, CONCURRENCY, DISABLING, SUSPEND, ORDER INDEPENDENCE AND CHOICE

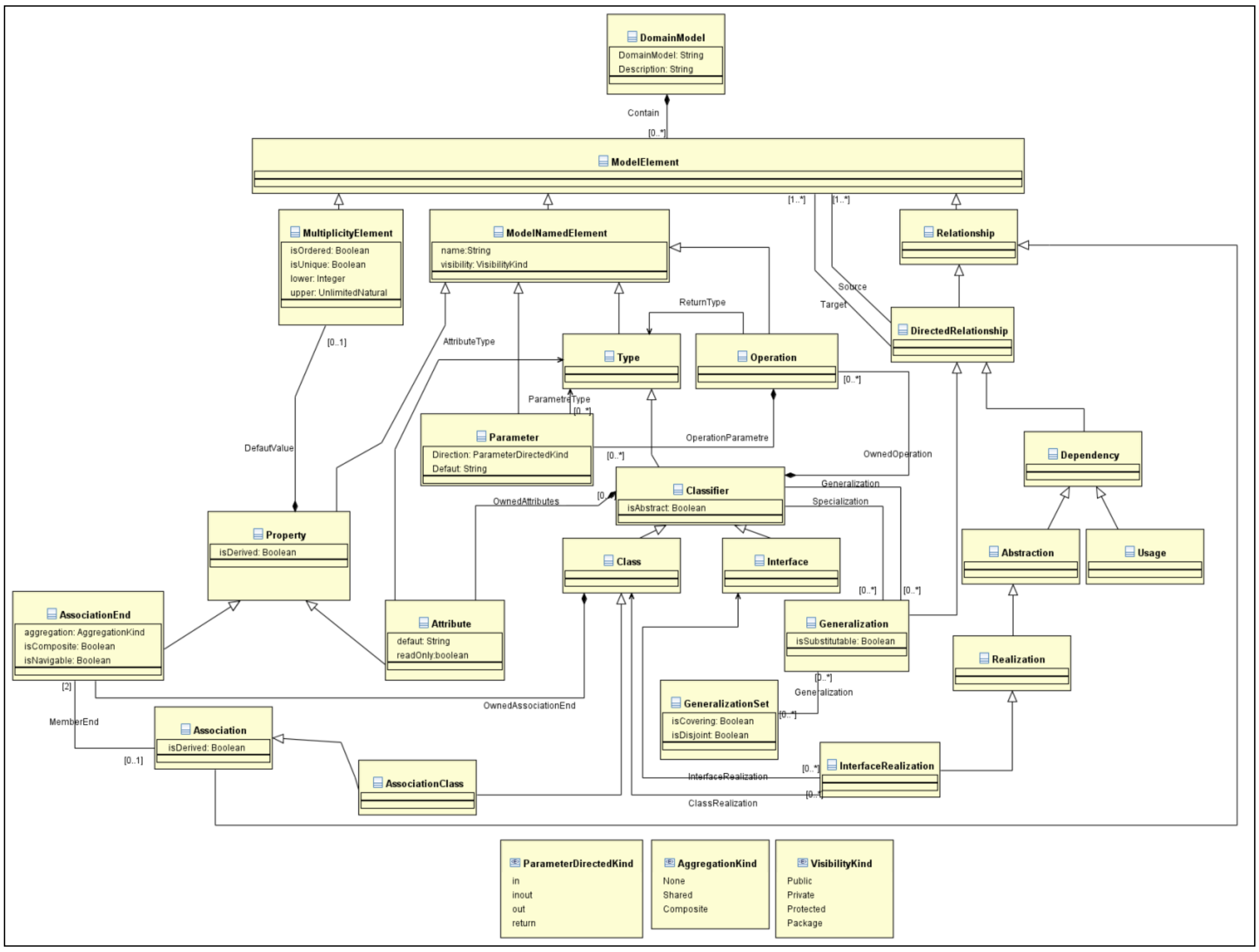
3.4. Domain

The UsiXML domain model describes the various entities manipulated by a user while interacting with the system [UsiXML07, Sta08]. This model specifies the main concepts of a User Interface by identifying the relationships among all the entities within the scope of the User Interface, their attributes and the methods encapsulated within the entities. As such, the domain model can be modeled with an object-oriented model like the UML class diagram [OMG08]. A class diagram of UML is a type of static structure diagrams that provides a good expressiveness to describe the structure of a system by using the classes, their attributes, and the relationships between the classes [Seo06]. For this reason, the UsiXML domain model uses the UML class diagram to describe the different entities manipulated through a User Interface.

In the next section, we describe the UsiXML domain meta-model which is based on the UML class diagram meta-model [OMG09].

3.4.1. Overview

Figure 6 shows the UsiXML domain meta-model adopted from the UML class diagram. The UML Class is the main concept of this meta-model. It describes a User Interface (UI) entity, its attributes and its operations. A UI entity attribute and an operation are represented respectively by the property class and operation class. In turn, the relationship between entities can be an association that describes a semantic relationship between entities or a generalization that describes a taxonomic relationship



between them.

Figure 6 - UsiXML domain Meta-model adopted from the UML Class diagram [OMG09]

The meta-model depicted in Figure 1 is a simplified view of the meta-model of UML 2.2 described by OMG specification in [OMG09]. The classes of this meta-model are defined in the following section. Note that, the definition of the main classes of the meta-model is from the [OMG09]. Another note is that, the class Property, described in [OMG09], is specialized into two classes: Attribute and an Association End in order to provide a more understandable meta-model.

- *Abstraction:* An abstraction is a relationship that relates two elements or sets of elements that represent the same concept at different levels of abstraction or from different viewpoints.
  - *Attributes:*

    No specific attribute

- *AggregationKind:* is an enumeration type that specifies the kind of aggregation of a property. AggregationKind is an enumeration of the following values:

  • none: Indicates that the property has no aggregation.

  • shared: Indicates that the property has a shared aggregation.

  • composite: Indicates that the property is aggregated compositely, i.e., the composite object has responsibility for the existence and storage of the composed objects (parts).

- *AssociationClass:*. An AssociationClass can be seen as an association that also has class properties, or as a class that also has association properties.
  - *Attributes:*

    No specific attribute

- *AssociationEnd:* This class represents the role played by one class within an association.
  - *Attributes*

    • aggregation (**AggregationKind**): Specifies the kind of aggregation that applies to the association. The default value is none.

    • isComposite (**Boolean**) : This is a derived value, indicating whether the aggregation of the association is composite or not.

    • navigable (**Boolean**) : This is a derived value, indicating whether the association is navigable or not.

- *Association:* An association specifies a semantic relationship that can occur between typed instances. It has at least two members end represented by properties, each of which is connected to the type of the end.
  - *Attributes*

    • isDerived **(Boolean):** Specifies whether the association is derived from other model elements such as other associations. The default value is false.

- *Attribute:* This class represents an attributes of a class or an interface.
  - *Attributes:*

    • default **(String)**: A String that is evaluated to give a default value for the attribute when an object of the owning Class is instantiated.

    • isReadOnly **(Boolean):** If true, the attribute may only be read, and not written. The default value is false.

- *Class:* A class describes a set of objects (entities) that share the same specifications of features, and semantics.
  - *Attributes:*

• name (**String**) : The name of the class.

• isAbstract (**Boolean**) : If this attribute is true, then the class does not provide a complete declaration and can typically not be instantiated. Default value is false.

• visibility (**VisibilityKind**) : Determines the class accessibility.

- *Classifier:* A classifier is an abstract class that describes the common elements of interfaces and classes.
  - *Attributes:*

    • isAbstract (**Boolean**) : If this attribute is true, then the class does not provide a complete declaration and can typically not be instantiated. Default value is false.

- *Dependency:* A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation.
  - *Attributes:*

    No specific attribute

- *DirectedRelationship:* A directed relationship references one or more source elements and one or more target elements.
  - *Attributes:*

    No specific attribute

- *DomainModel:* Domain model is a description of the classes of objects manipulated by a user while interacting with a system. A domain model is made up of submodels.
  - *Attributes*

    • domainName **(String):** the name of the domain model.

    • description **(String):** the textual description of the domain model.

- *Generalization:* A generalization is a taxonomic relationship between a more general class and a more specific class. Each instance of the specific class is also an indirect instance of the general class. Thus, the specific class inherits the features of the more general class.
  - *Attributes*

    • isSubstitutable **(Boolean):** Indicates whether the specific class can be used wherever the general class can be used. If true, the execution traces of the specific class will be a superset of the execution traces of the general class. The default value is true.

- *GeneralizationSet:* *A* GeneralizationSet defines a particular set of Generalization relationships that describe the way in which a general class (or superclass) may be divided using specific subtypes. For example, a GeneralizationSet could define a partitioning of the class Person into two subclasses: Male Person and Female Person.
  - *Attributes:*
    - isCovering (**Boolean**): Indicates whether or not the set of specific classes are covering for a particular general class. When isCovering is true, every instance of a particular general Class is also an instance of at least one of its specific Classes for the GeneralizationSet. When isCovering is false, there are one or more instances of the particular general Class that are not instances of at least one of its specific Classes defined for the GeneralizationSet.

- isDisjoint **(Boolean)** : Indicates whether or not the set of specific classes in a Generalization relationship have instance in common. If isDisjoint is true, the specific Classes for a particular GeneralizationSet have no members in common; that is, their intersection is empty. If isDisjoint is false, the specific Classes in a particular GeneralizationSet have one or more members in common; that is, their intersection is not empty.

➢ *Interface:* An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. Since interfaces are declarations, they are not instantiable. Instead, an interface specification is implemented by an instance of an instantiable class, which means that the instantiable class presents a public facade that conforms to the interface specification

- o *Attributes:*

  No specific attribute

➢ *InterfaceRealization:* An InterfaceRealization is a specialized Realization relationship between a class and an Interface. This relationship signifies that the realizing class conforms to the contract specified by the Interface.

- o *Attributes:*

  No specific attribute

➢ *ModelElement:* A Model element is a constituent of a domain model. This class represents an abstract generalization class that is specialized in the meta-model.

- o *Attributes*

  No specific attribute

➢ *ModelNamedElement:* A named element represents elements that may have a name. This class represents an abstract generalization class that is specialized in the meta-model.

- o *Attributes:*

  • name (**String**) : The name of the element.

  • visibility (**VisibilityKind**) : Determines the element accessibility.

➢ *MultiplicityElement:* A multiplicity Element class is represent an inclusive interval of non-negative integers beginning with a lower bound and ending with a (possibly infinite) upper bound

- o *Attributes*

  • isOrdered (**Boolean**) : For a multivalued multiplicity, this attribute specifies whether the values in an instantiation of this element are sequentially ordered. Default is false.

  • isUnique (**Boolean**) : For a multivalued multiplicity, this attributes specifies whether the values in an instantiation of this element are unique. Default is true.

  • lower (**Integer**) : Specifies the lower bound of the multiplicity interval, if it is expressed as an integer.

  • upper (**UnlimitedNatural**) : Specifies the upper bound of the multiplicity interval, if it is expressed as an unlimited natural.

➢ *Operation:* An operation specifies the name, type, and the parameters, of a class operation

- o *Attributes*

  • isOrdered **(Boolean):** Specifies whether the return parameter is ordered or not, if present.

  • isUnique **(Boolean):** Specifies whether the return parameter is unique or not, if present.

  • lower (**Integer**): Specifies the lower multiplicity of the return parameter, if present.

- upper (**UnlimitedNatural**) : Specifies the upper multiplicity of the return parameter, if present. This is derive

- *Parameter:* is a specification of an argument used to pass information into or out of an invocation of a operation.
  - *Attributes*
    - defaut **(String):** Specifies a String that represents a value to be used when no argument is supplied for the Parameter.
    - direction **(ParameterDirectionKind):** Indicates whether a parameter is being sent into or out of a operation

- *ParameterDirectionKind:* Parameter Direction kind is an enumeration type that defines literals used to specify direction of parameters. Parameter Direction kind is an enumeration of the following values:
  - In: indicates that parameter values are passed into the behavioral element by the caller.
  - Inout: indicates that parameter values are passed into a behavioral element by the caller and then back out to the caller from the behavioral element.
  - Out: indicates that parameter values are passed from a behavioral element out to the caller.
  - Return: indicates that parameter values are passed as return values from a behavioral element back to the caller.
- *Property:* This class represents a common property of all the instances of a class, interface or an Association. This class is a generalization of an Attribute and an Association End
  - *Attributes:*
    - isDerived **(Boolean):** Specifies whether the Property is derived, i.e., whether its value or values can be computed from other information. The default value is false.

- *Realization:* Realization is a specialized abstraction relationship between two sets of model elements, one representing a specification and the other represents an implementation of the latter.
  - *Attributes:*

    No specific attribute

- *Relationship:* A relationship references one or more related elements.
  - *Attributes:*

    No specific attribute

- *Type:* This class is used as a constraint on the range of values represented by a typed element. Type is an abstract generalization class.
  - *Attributes:*

    No specific attribute

- *VisibilityKind:* is an enumeration of the following values:
  - public
  - private

- protected
- package

- *Usage:* usage is a relationship in which one element requires another element (or set of elements) for its full implementation or operation.
  - *Attributes:*

    No specific attribute

3.4.2. Example

Figure 7 gives an example of a UsiXML domain model expressed using a UML class diagram. In this domain model, the various entities manipulated through a User Interface are represented by classes (e.g. Person, Flight, etc.). Each class can have a set of properties and set of operations (e.g. Person class has two properties: DepartureDate, and ArrivedDate, and two operations: OpenFlight and CloseFlight). In the UsiXML domain model, two kinds of relationships can exist between entities (classes): 1) Association relationship: it consists of two classes, each playing a specific role (e.g. a client has the role to make a reservation). Each role is characterized by a constraint on the number of instances of a class connected across an association (multiplicity). Note that, an association can represent a composition relation between a class and another one (e.g. an airport is composed of several terminals). Another note is the fact that, an association can have a property (e.g. an air

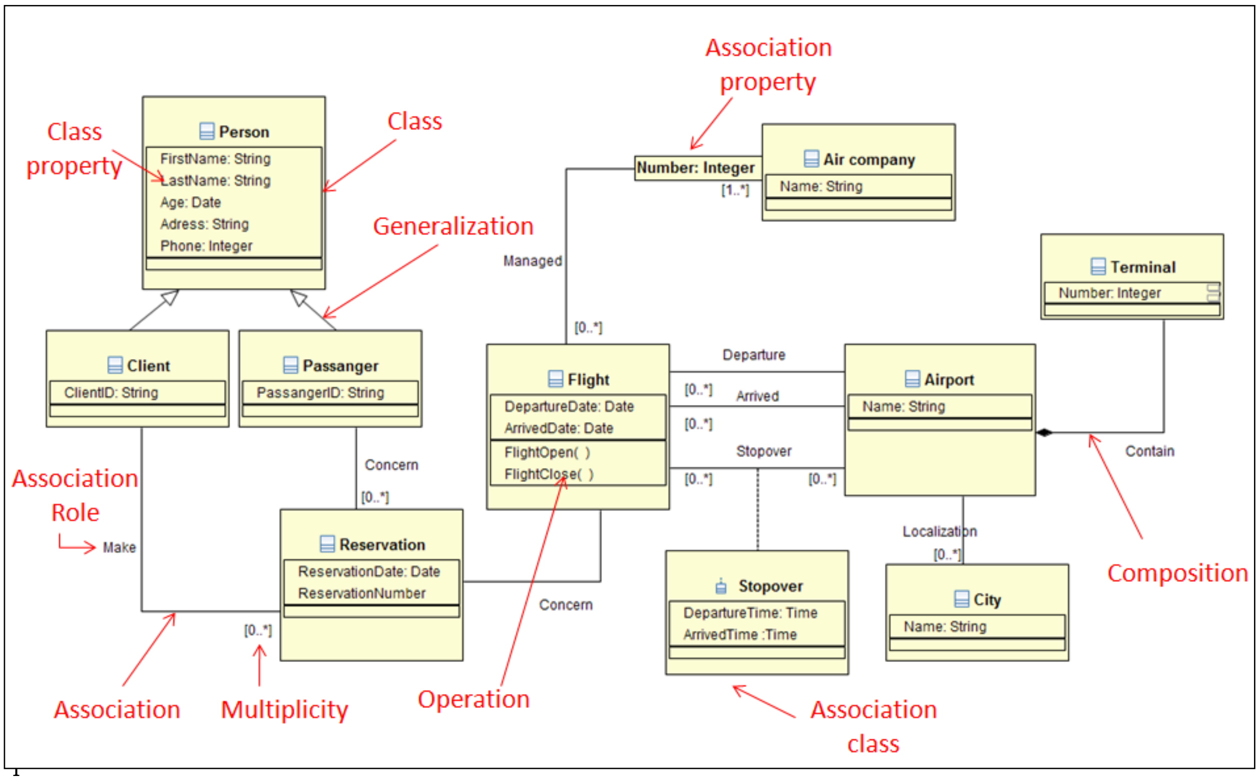


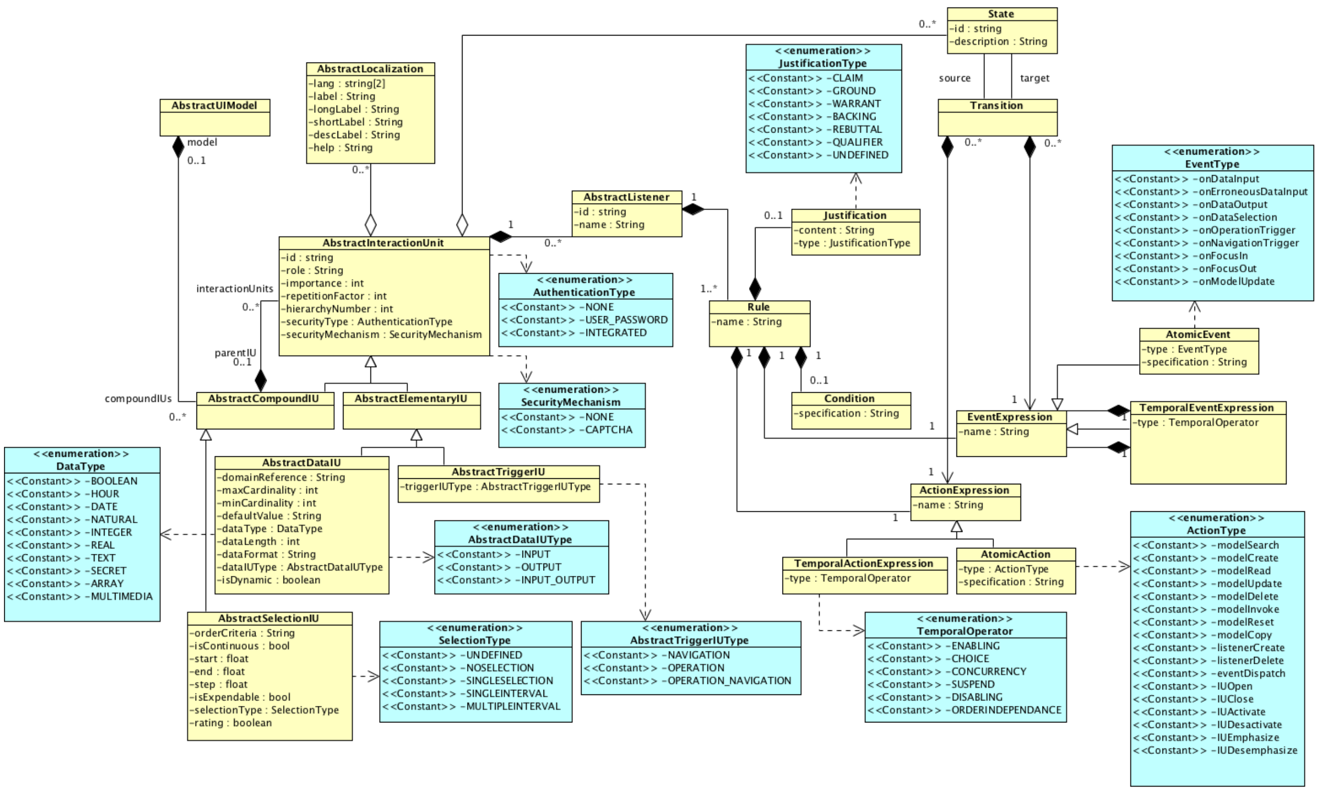
Figure 7 - UsiXML Domain Model Example

ght of a

specific number). This kind of association can be modeled as a class if the association has itself a set of properties (e.g. Class Stopover). 2) Generalization relationship: it links a super-class to one or several other sub-classes (e.g. the super class Person is a generalization of Client class and Passenger class).

3.5. Abstract User Interface

The Abstract User Interface (AUI) (corresponding to the Platform-Independent Model–PIM– in MDE) is an expression of the UI in terms of interaction spaces (or presentation units), independently of which interactors are available and even independently of the modality of interaction (graphical, vocal, haptic …). An interaction space is a grouping unit that supports the execution of a set of logically connected tasks.



The chosen modeling is to compose the *AbstractUIModel* by a set of *AbstractCompoundIUs* that are themselves sets of *AbstractInteractionUnit*: every abstract object is then such an interaction unit. Each interaction unit may have one or more *AbstractListeners*. These listeners allow defining the dynamic of the model, the way the interaction units will react to the different events. Additionally, the *AbstractInteractionUnits* are related to states and transitions enabling the possibility to comply with the State Chart XML (SCXML) defined by W3C. (http://www.w3.org/TR/scxml/)

- *AbstractUIModel*: This model describes canonically a user interface in terms of abstract interaction units, relationships and listeners in a way that is independent from the concrete interaction units available on the targets. The abstract user interface model is independent of the target device or modality.
- *AbstractInteractionUnit*: Main entity of the model, every abstract object is an *AbstractInteractionUnit*.
  - *Attributes:*
    - id **(String)**: identification string of the *AbstractInteractionUnit*.
    - role **(String)**: name of the role played by the *AbstractInteractionUnit*.
    - importance **(String)**: importance level of the *AbstractiInteractionUnit*.
      - To be discussed later, for the moment a 5 level scale can be used like "Very high"/"High"/"Medium"/"Low"/"Very low".
    - repetitionFactor **(int)**: number of times the *AbstractInteractionUnit* is repeated in the parent entity.
    - hierarchyNumber **(int)**: allows to order the *AbstractionInteractionUnit* in its parent entity, in function of the other *AbstractInteractionUnits* contained in the same parent.

- securityType **(AuthenticationType)**: defines the type of authentication for the *AbstractInteractionUnit.*
- securityMechanism **(SecurityMechanism)**: defines the type of security mechanism for the *AbstractInteractionUnit.*

- *AuthenticationType*
  - *Attributes*
    - NONE
    - USER_PASSWORD
    - INTEGRATED
      - Stands for an authentication that can be done by the system, not by the user. For example, through a MAC address or a certificate.
      -
- *SecurityMechanism*
  - *Attributes*
    - NONE
    - CAPTCHA

- *AbstractLocalization*: Describes text attributes for *AbstractInteractionUnits*. It is useful for internationalization.
  - *Attributes*
    - lang **(String[2])**: international code (2 letters) for the language supported by the *AbstractLocalization*.
    - label **(String)**: description label of the *AbstractInteractionUnit.*
      - Example: "department".
    - longLabel **(String)**: label as it appears in the interface.
      - Example: "Department".
    - shortLabel **(String)**: short version of the label.
      - Example: acronym, like "dept".
    - descLabel **(String)**: description label of the *AbstractInteractionUnit.*
      - Example: "the name of the department".
    - help **(String)**: textual help provided specifically for the *AbstractInteractionUnit*.

- *AbstractCompoundIU:* Composition of one or several *AbstractInteractionUnit.*
- *AbstractSelectionIU:* Special type of *AbstractCompoundIU* representing a way to interact with the interface by selecting an item in a list.
  - *Attributes:*
    - orderCriteria **(String)**: criteria used to sort the selection.
      - Example: "alphabetical"
    - isContinuous **(Boolean)**: specifies if the selection is continuous.
      - Example: A selection that may be specialized at concrete level as a voice selection among any number between 0 and 1.
    - start **(Float)**: starting number (for numerical selection).
    - end **(Float)**: ending number (for numerical selection).

- step **(Float)**: interval between two successive items, by starting by *start* and ending by *end* (for numerical selection).
- isExpandable **(Boolean)**: specifies if the user can add item in the selection.
- selectionType **(SelectionType)**: specifies the type of selection of the unit.
- Rating **(Boolean)**: specifies if the unit is used for a rating.

- *SelectionType*
  - *Attributes*
    - UNDEFINED
    - NOSELECTION
    - SINGLESELECTION
      - One item in the list is selectable.
    - SINGLEINTERVAL
      - An interval of the list is selectable.
        - Example: for a list of digit between 0 and 9, an interval may be the digits between 2 and 5.
    - MULTIPLEINTERVAL
      - Multiple intervals of the list are selectable.
        - Example: for a list of digit between 0 and 9, an interval may be the digits between 2 and 5 and a second one between 7 and 8.
- *AbstractElementaryIU:* Atomic *AbstractInteractionUnit* that can be of 2 types: *AbstractDataIU* or *AbstractTriggerIU.*
- *AbstractDataIU:* Interaction unit allowing to link data from the Domain Model with elements of the abstract user interface.
  - *Attributes:*
    - domainReference **(String)**: reference allowing to link the Domain Model in order to populate the *AbstractDataItem*.
    - maxCardinality **(Integer)**: maximum number of items in the *AbstractDataIU.*
    - minCardinality **(Integer)**: minimum number of items in the *AbstractDataIU.*
    - defaultValue **(String)**: default value or concatenation of default values that can be used if the required value is not available in the Domain Model.
    - dataType **(DataType)**: type of the data.
    - dataLength **(Integer)**: size of the data, in bytes.
    - dataFormat **(String)**: format of the data.
      - Example: .doc, .pdf, .xml, …
    - dataIUType **(AbstractDatayIUType)**: type of interaction.
    - isDynamic **(Boolean)**: specifies if the content can evolve in the time.
- *DataType*
  - *Attributes*
    - BOOLEAN
    - HOUR
    - DATE
    - NATURAL

        - INTEGER
        - REAL
        - TEXT
        - SECRET
            - Stands for a type of data that we want to keep secret.
        - ARRAY
        - MULTIMEDIA

- *AbstractDataIUType*
    - *Attributes*
        - INPUT
        - OUTPUT
        - INPUT_OUTPUT
- *AbstractTriggerIU:* Interaction unit allowing triggering an event.
    - *Attributes*
        - triggerIUType **(AbstractTriggerIUType)**: type of event triggered.
- *AbstractTriggerIUType*
    - *Attributes*
        - NAVIGATION
        - OPERATION
        - OPERATION_NAVIGATION

- *AbstractListener:* Entity used to describe the behavior of the *AbstractInteractionUnit* by using Event-Conditio-Action (ECA) rules.
    - *Attributes*
        - id **(String)**: identification string of the *AbstractListener.*
        - name **(String)**: name of the *AbstractListener.*
- *Rule*
    - *Attributes*
        - name **(String)**: name of the *AbstractListener.*
- *Justification:* The justification is a kind of motivation for the ECA rules, it is not used for describing the interface itself but more for documentation purposes.
    - *Attributes*
        - content **(String)**: text representing the justification itself.
        - justificationType **(JustificationType)**: type of justification, the different possibilities are described in the *AbstractJustificationType* description.
- *JustificationType*
    - *Attributes*
        - CLAIM: assertion put forward publicly for general acceptance with the implication that there are underlying 'reasons' that could show it to be 'well founded' and therefore entitled to be generally accepted.
            - Example: "Our organization should cut costs next quarter"

- GROUND: the term 'ground' refers to the specific facts relied on to support a given claim.
  - Example: "Our organization has lost money the last 3 quarters"
- WARRANT: assertion that entitles you to interpret or link the grounds (facts) as support of the claim.
  - Example: "When losing money, organizations should cut expenses as much as possible"
- BACKING: the 'backing' consists of a very general set of background assumptions which, in effect, legitimize the basis for believing in the warrant. That is, if the warrant is not accepted on its surface, then the backing is called into play to add deeper support to the argument.
  - Example: "A business can't continue to lose money and stay in business"
- REBUTTAL: the 'rebuttal' presents the possible exceptions or objections as to why the claim, the grounds, the warrants, or the backing may not hold for the situation under discussion.
  - Example: "That may be true in general, but not with the customers of our company. Besides that, times have changed; the economy as changed; the dollar has fallen in value.
- QUALIFIER: word that indicates how strongly the claim is being asserted, or how likely that something might occur.
  - Example: "probably", "certainly", "very likely", etc.
- UNDEFINED

➢ *EventExpression:* specifies a set of events with relationships between them. The events are represented by *AtomicEvents* and are related by *TemporalEventExpressions*.

- *Attributes*
  - name **(String)**: name of the *EventExpression.*

➢ *AtomicEvent*

- *Attributes*
  - type **(EventType)**: type of event, the different possibilities are described in the *EventType* description.
  - Specification **(String)**: kind of argument that, used in conjunction with type, allows specifying Event to listen.

➢ *EventType:*

- *Attributes*
  - onDataInput: new data has been entered by the user
  - onErroneousDataInput: new erroneous data has been entered by the user
  - onDataOutput: new data has been output through the interface
  - onDataSelection: some data has been selected by the user
  - onOperationTrigger: an *AbstractOperationIU* has been activated
  - onNavigationTrigger: an *AbstractNavigationIU* has been activated
  - onIUFocusIn: an *AbstractInteractionUnit* has been focused in
  - onIUFocusOut: an *AbstractInteractionUnit* has been focused out
  - onModelUpdate: the Domain Model has been updated

➢ *TemporalEventExpression*

- *Attributes*
  - type **(TemporalOperator)**: type of relationship involved between two *AtomicEvents*, the different possibilities are described in the *TemporalOperator* description.

➢ *EventType:* (see Task MM for description of the attributes)

- *Attributes*

  - ENABLING
  - CHOICE
  - CONCURRENCY
  - SUSPEND
  - DISABLING
  - ORDERINDEPENDANCE

- Condition: logical test that, if satisfied or evaluates to true, causes the action to be carried out.
  - *Attributes*
    - specification **(String)**: logical formula representing the condition to evaluate, for the moment under the form of a String.
- ActionExpression: specifies a set of actions with relationships between them. The actions are represented by *AtomicActions* and are related by *TemporalActionExpressions*.
  - *Attributes*
    - name **(String)**: name of the *ActionExpression.*
- AtomicAction
  - *Attributes*
    - type **(ActionType)**: type of action, the different possibilities are described in the *ActionType* description.
    - specification **(String)**: kind of argument that, used in conjunction with type, allows specifying the action to launch.
      - Example: for an *AbstractActionType* 'modelUpdate', the actionSpecification could specify which field to change and what is the new value.
- AbstractActionType
  - *Attributes*
    - modelSearch: search for model elements based on logical formula
    - modelCreate: create a new model in the Domain Model
    - modelRead: read a specified field in the Domain Model
    - modelUpdate: update a field in the Domain Model
    - modelDelete: delete a field in the Domain Model
    - modelInvoke: perform a query external to the current Domain Model
    - modelReset: reset the Domain Model with the initial parameters
    - modelCopy: copy the Domain Model
    - listenerCreate: create a new listener on a specified *AbstractInteractionUnit*
    - listenerDelete: delete a listener of a specified *AbstractInteractionUnit*
    - eventDispatch: dispatch the event to another *AbstractInteractionUnit*
    - IUOpen: open a specified *AbstractInteractionUnit*
    - IUClose: close a specified *AbstractInteractionUnit*
    - IUActivate: activate a specified *AbstractInteractionUnit*
    - IUDesactivate: desactivate a specified *AbstractInteractionUnit*
    - IUEmphasize: focus in a specified *AbstractInteractionUnit*
    - IUDesemphasize: focus out a specified *AbstractInteractionUnit*
- TemporaActionExpression

- *Attributes*
  - type **(TemporalOperator)**: type of relationship involved between two *AtomicActions*, the different possibilities are described in the *TemporalOperator* description.

➢ State: possible state of an *AstractInteractionUnit*. An *AbstractInteractionUnit* may have many different states.

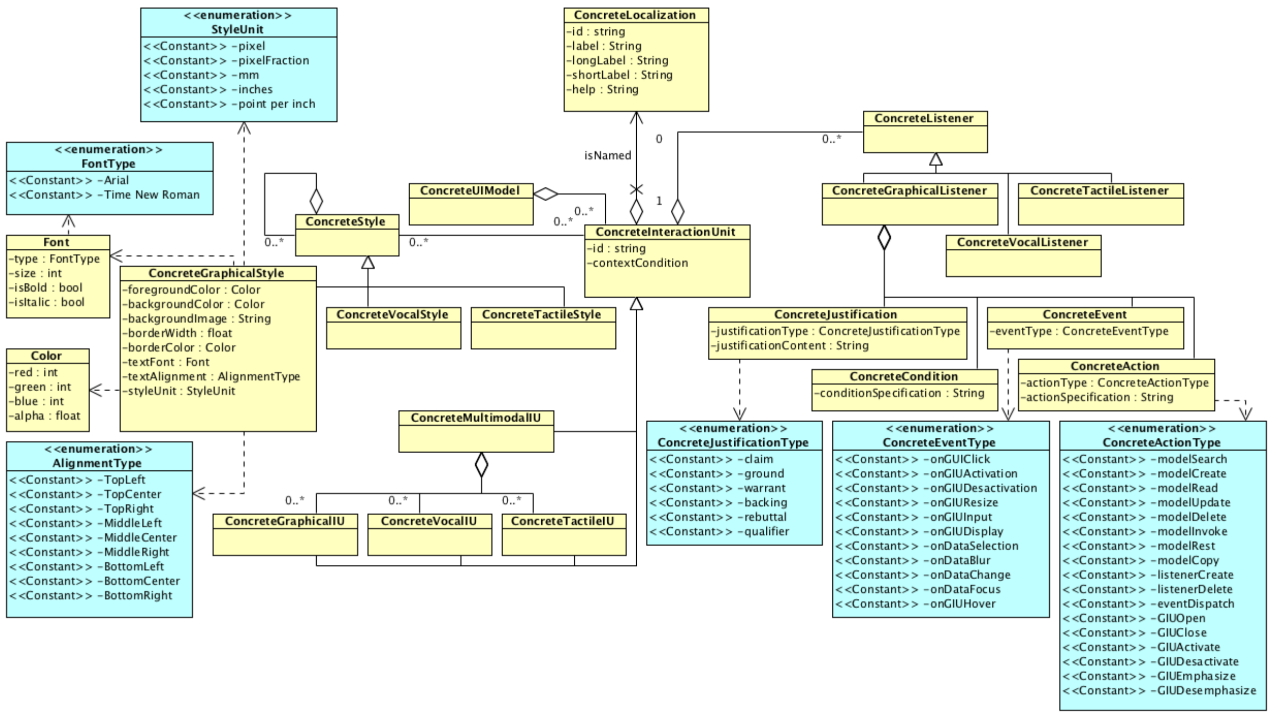
- *Attributes*
  - id **(String)**: identification string of the state.
  - Description **(String)**: description of the state.

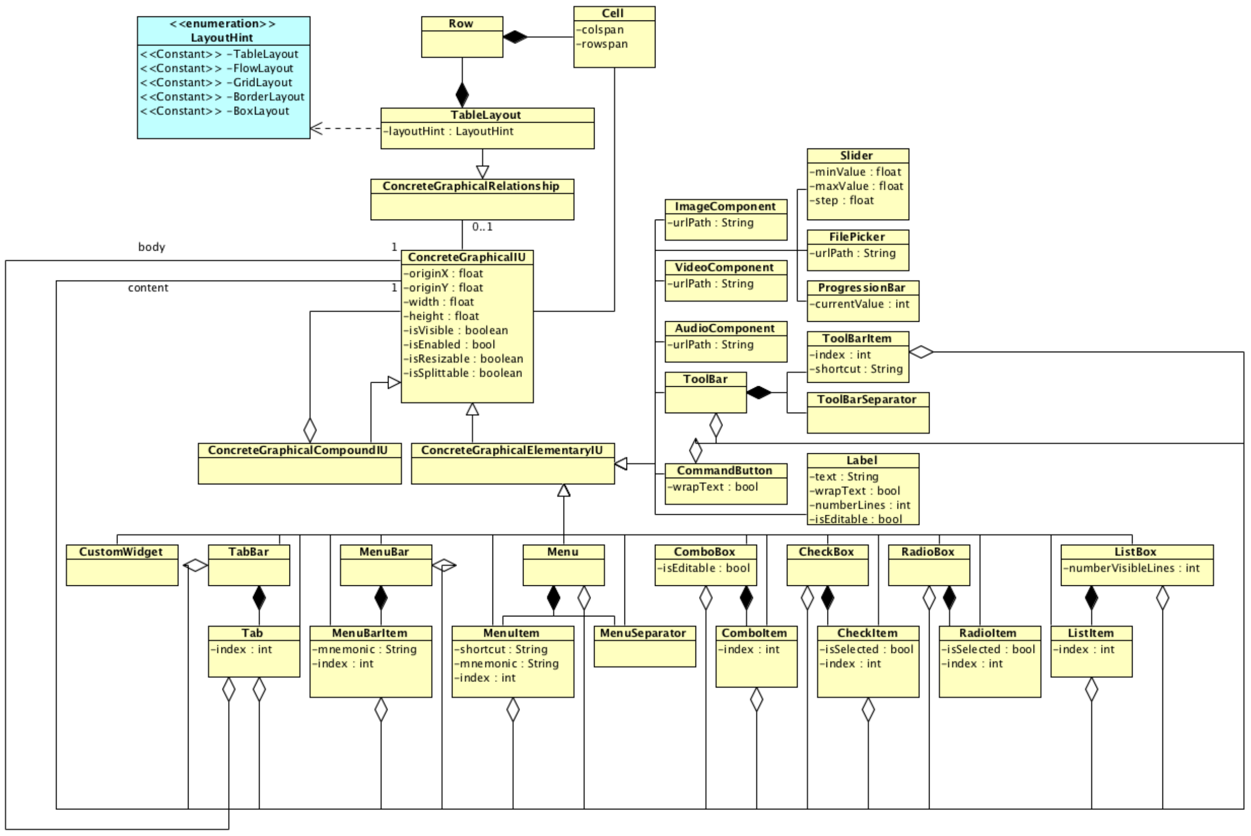
➢ Transition: is related to *State* by two relationships meaning that a transition has source state and a target state. Additionally it is related to *EventExpression* and *ActionExpression*.

3.6. Concrete User Interface

The Concrete User Interface (CUI) (corresponding to the Platform-Specific Model–PSM– in MDE) is an expression of the UI in terms of "concrete interaction units", which depends on the type of platform and media available and has a number of attributes that define more concretely how the user should perceive it. "Concrete interaction units" are, in fact, an abstraction of actual UI components generally included in toolkits.

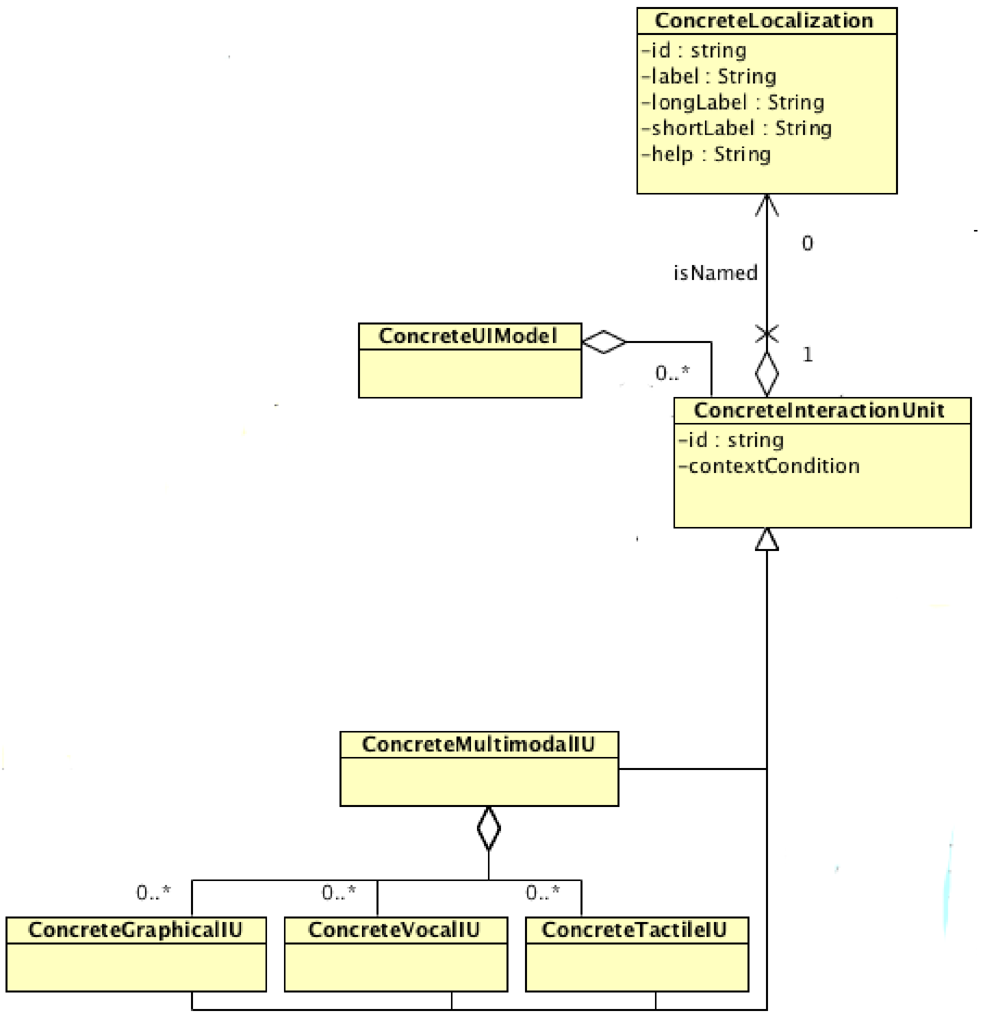
3.6.1. Overview





The chosen modeling is to compose the *ConcreteUIModel* by a set of *ConcreteInteractionUnit*: every concrete object is then such an interaction unit. These interaction units may own *ConcreteStyles* allowing describing a style for each concrete interaction unit. Furthermore, each interaction unit may have one or more *ConcreteListeners*. These listeners allow defining the dynamic of the model, the way the interaction units will react to the different events. The goal of the Concrete User Interface model is to specify the modality. For the moment, only the graphical modality is described.

3.6.2. Main entities

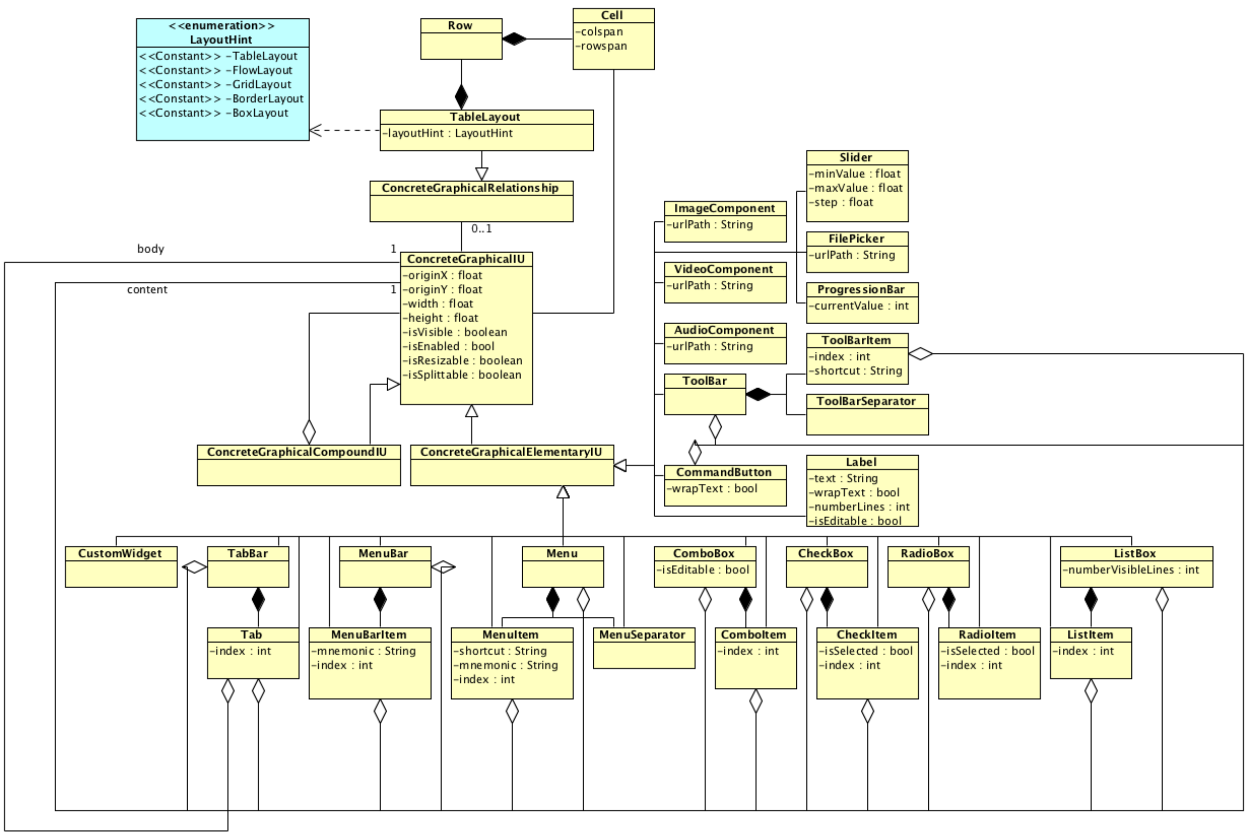


- *ConcreteUIModel*: Last level of abstraction (before the code). Describes user interfaces independently of the toolkit or the execution environment (Java-Swing, Web, VoiceXML …). The modality is known.
- *ConcreteInteractionUnit*: Main entity of the model, every concrete object is a *ConcreteInteractionUnit*. For the moment, it is refined in three simple modalities (graphical, vocal, tactile) and compound modality (multimodal) and only the graphical modality is refined (see *ConcreteGraphicalIU* description).
    - *Attributes:*
        - id **(Integer)**: Identification number of the *ConcreteInteractionUnit*.
        - contextCondition : Link to the Context model.
- *ConcreteLocalization*: Allows specifying locales in different languages for the concrete interaction unit.
    - *Attributes:*
        - id **(Integer)**: Identification number of the *ConcreteLocalization*.
        - label **(String)**: Description label of the *ConcreteInteractionUnit*.
            - Example: “department”.
        - longLabel **(String)**: Long version of the label.
            - Example: “the name of the department”.
        - shortLabel **(String)**: short version of the label.

- Example: acronym, like “dept”.

- help **(String)**: textual help provided specifically for the *ConcreteInteractionUnit*.

3.6.3. ConcreteGraphicalIU



➢ *ConcreteGraphicalIU:* Main graphical entity of the model, every graphical entity is a *ConcreteGraphicalIU*. It can be of two types: *ConcreteGraphicalCompoundIU* or *ConcreteGraphicalElemenatryIU*. The elementary entities represent the atomic graphical widgets while the compound entities behave as containers gathering other compound entities and elementary entities. The way the entities are organized is described in the *ConcreteGraphicalRelationship*.

- *Attributes*
  - originX **(Float)**: x coordinate of the origin point of the *ConcreteGraphicalIU*.
  - originY **(Float)**: y coordinate of the origin point of the *ConcreteGraphicalIU*.
  - width **(Float)**: width of the *ConcreteGraphicalIU*.
  - height **(Float)**: height of the *ConcreteGraphicalIU*.
  - isVisible **(Boolean)**: specifies if the *ConcreteGraphicalIU* is visible.
  - isEnabled **(Boolean)**: specifies if the *ConcreteGraphicalIU* is enabled.
  - isResizable **(Boolean)**: specifies if the *ConcreteGraphicalIU* is resizable.
  - isSplittable **(Boolean)**: specifies if the *ConcreteGraphicalIU* is splittable.

➢ *ConcreteGraphicalCompoundIU:* Entity allowing to gather combinations of other *ConcreteGraphicalCompoundIU* and *ConcreteGraphicalEmentaryIU*.

➢ *ConcreteGraphicalElementaryIU:* Atomic graphical entity.

➢ *TabBar:* Bar gathering tab entries to click in order to show the tab.

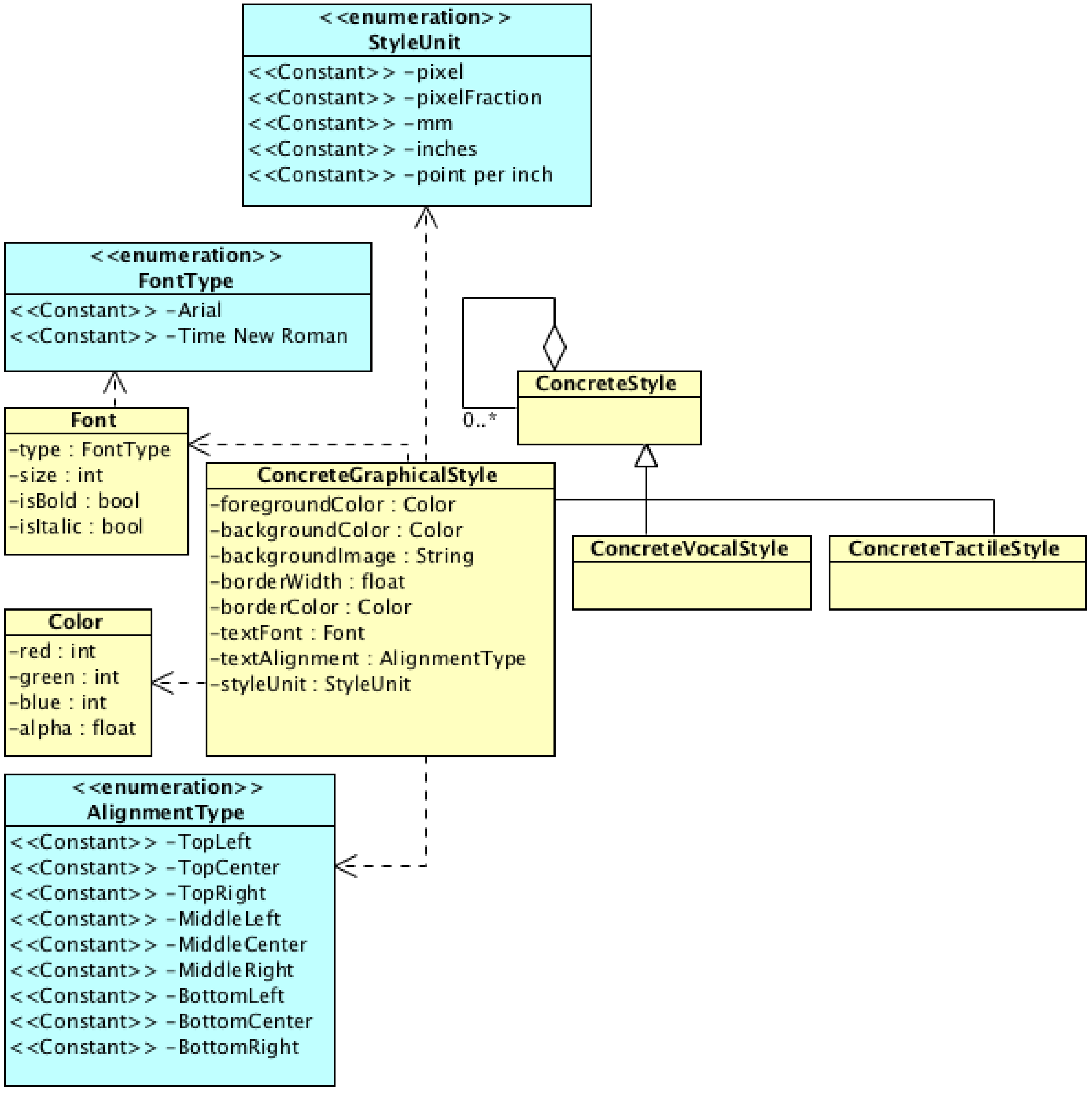
- *Tab:* Entity linking the tab bar entry and the content of the Tab (that is a *ConcreteGraphicalCompoundIU*).
  - *Attributes*
    - text **(String)**: text to display as entry for the tab in the tab bar.
    - Index **(Integer)**: index of the tab in the tab bar.
- *MenuBar:* Bar gathering different menu bar items.
- *MenuBarItem:* Item linking an item of the menu bar with a menu.
  - *Attributes*
    - mnemonic **(String)**: combination of touches to press in order to uncollapse the associated menu.
    - index **(Integer)**: Index of the menu bar item in the menu bar.
- *Menu:* Organizer of separators and items allowing launching specific menu actions.
- *MenuItem:* Item allowing launching specific menu action.
  - *Attributes*
    - shortcut **(String)**: combination of touches to press in order to launch the menu item action.
    - mnemonic **(String)**: combination of touches to press in order to launch the menu item action after having uncollapsed the parent menu.
    - index **(Integer)**: Index of the menu item in the menu.
- *MenuSeparator:* Separator to be used in menus.
- *ComboBox:* Combination of a text label and a list box: only one entry is visible and a button on the right allows showing the entire list for selection.
  - *Attributes*
    - isEditable **(Boolean)**: specifies if the user can type its own entry.
- *ComboItem:* Item of the combo box list.
  - *Attributes*
    - index **(Integer)**: Index of the combo item in the combo box.
- *CheckBox:* Entity allowing the user to make multiple selections from a number of options.
- *CheckItem:* Item representing an option of the check box.
  - *Attributes*
    - isSelected **(String)**: specifies if the current item is selected.
    - index **(Integer)**: Index of the check item in the check box.
- *RadioBox:* Entity allowing the user to make single selection from a number of options.
- *RadioItem:* Item representing an option in the radio box.
  - *Attributes*
    - isSelected **(String)**: specifies if the current item is selected. Setting this attribute to 'true' causes setting to false the isSelected attribute of all the other radio items of the radio box.
    - index **(Integer)**: Index of the radio item in the radio box.
- *ListBox:* Entity allowing to select one or more items from a list.
- *ListItem:* Item of the list box.
  - *Attributes*
    - index **(Integer)**: Index of the list item in the list.
- *ToolBar:* Bar allowing organizing buttons and separators for specific commands that are not belonging to the menu.

- *ToolBarItem:* Item of the toolbar.
    - *Attribute*
        - index **(Integer)**: Index of the toolbar item in the toolbar.
        - shortcut **(String)**: combination of touches to press in order to launch the toolbar button action.
- *ToolBarSeparator:* Separator to be used in toolbars.
- *CommandButton:* Free button allowing to launch a command.
    - *Attributes*
        - wrapText **(Boolean)**: specifies if the button has to automatically resize in order to wrap the text.
- *Label:* Simple component allowing to insert or display text.
    - *Attributes*
        - text **(String)**: text of the label.
        - wrapText **(Boolean)**: specifies if the label has to automatically resize in order to wrap the text.
        - numberLines **(Integer)**: maximum number of lines of the label (0 for infinite)
        - isEditable **(Boolean)**: specifies if the text can be changed.
- *ImageComponent:* Component allowing displaying an image.
    - *Attributes*
        - urlPath **(String)**: path to the image.
- *VideoComponent:* Component allowing playing a video.
    - *Attributes*
        - urlPath **(String)**: path to the video.
- *AudioComponent:* Component allowing playing an audio file.
    - *Attributes*
        - urlPath **(String)**: path to the audio file.
- *Slider:* Bar with a cursor that the user can move in order to specify a value.
    - *Attributes*
        - minValue **(Float)**: beginning value of the bar.
        - maxValue **(Float)**: ending value of the bar.
        - step **(Float)**: **)**: interval between two successive values of the bar, by starting by *minValue* and ending by *maxValue*).
- *FilePicker:* Component allowing uploading a file.
    - *Attributes*
        - urlPath **(String)**: path to the file.
- *ProgressionBar:* Bar that begins to 0 and finishes to 100 in order to represent a progression in percentages.
    - *Attributes*
        - currentValue **(Integer)**: value between 0 and 100 representing a percentage.
- *ConcreteGraphicalRelationship:* Entity linked to a *ConcreteGraphicalIU* in order to describe the way it is organized. If no *ConcreteGraphicalRelationship* is specified, then the organization is the following:
        - For a *ConcreteGraphicalCompoundIU*: the components contained in the *ConcreteGraphicalCompoundIU* are organized by placing their top-left corner to their specified origin, by taking the origin and the size/height of the considered *ConcreteGraphicalCompoundIU* as space reference.

    - For a *ConcreteGraphicalElementaryIU*: the items of the bars (*TabBar, MenuBar, ToolBar*) are organized from left to right, the items of boxes (*ComboBox, CheckBox, RadioBox, ListBox)* and *Menu* for top to bottom, by following the indexes.

- *TableLayout:* Defines the layout of the ConcreteGraphicalIU.
  - *Attributes*
    - layoutHint **(LayoutHint)**: Gives a hint on how the layout is structured.
- *Row:* If *TableLayout* hint is chosen, represents a row in the table.
- *Cell:* If *TableLayout* hint is chosen, represents a cell in the row of the table. This entity encloses a ConcreteGraphicalIU.
- *LayoutHint*
  - *Attribute*
    - TableLayout: Allows organizing in rows and cells, like in HTML.
    - FlowLayout: Allows organizing the content from left to right by specifying a justification.
    - GridLayout: Allows organizing the content as a table with cells of same sizes. By following their indexes, the components are added in the cells by left to right, by beginning at the top line until the bottom line.
    - BorderLayout: Allows organizing the content in 5 zones: North, East, South, West, and center.
    - BoxLayout: Allows to organize the content in a box of one line, where the direction of the line can be specified.

3.6.4. ConcreteStyle



- *ConcreteStyle:* Allows to specify a style for each *ConcreteInteractionUnit*. For the moment, it is refined in three simple modalities (graphical, vocal, tactile) and only the graphical modality is refined (see *ConcreteGraphicalStyle* description).
- *ConcreteGraphicalStyle:* Entity allowing specifying the style of a Concrete*GraphicalIU*.
  - *Attributes*
    - foregroundColor **(Color)**: color of the foreground.
    - backgroundColor **(Color)**: color of the background.
    - backgroudImage **(String)**: url to the image for the background.
    - borderWidth **(Float)**: width of the border (0 for no border).
    - borderColor **(Color)**: color of the border.
    - textFont **(Font)**: font of the text, if text is present.
    - textAlignment **(AlignmentType)**: alignment of the text, if text is present.
    - styleUnit **(StyleUnit)**: style of the units used for measuring sizes.
- *StyleUnit:*
  - *Attibutes*

- Pixel
- PixelFraction
- Mm
- Inches
- Point per inch

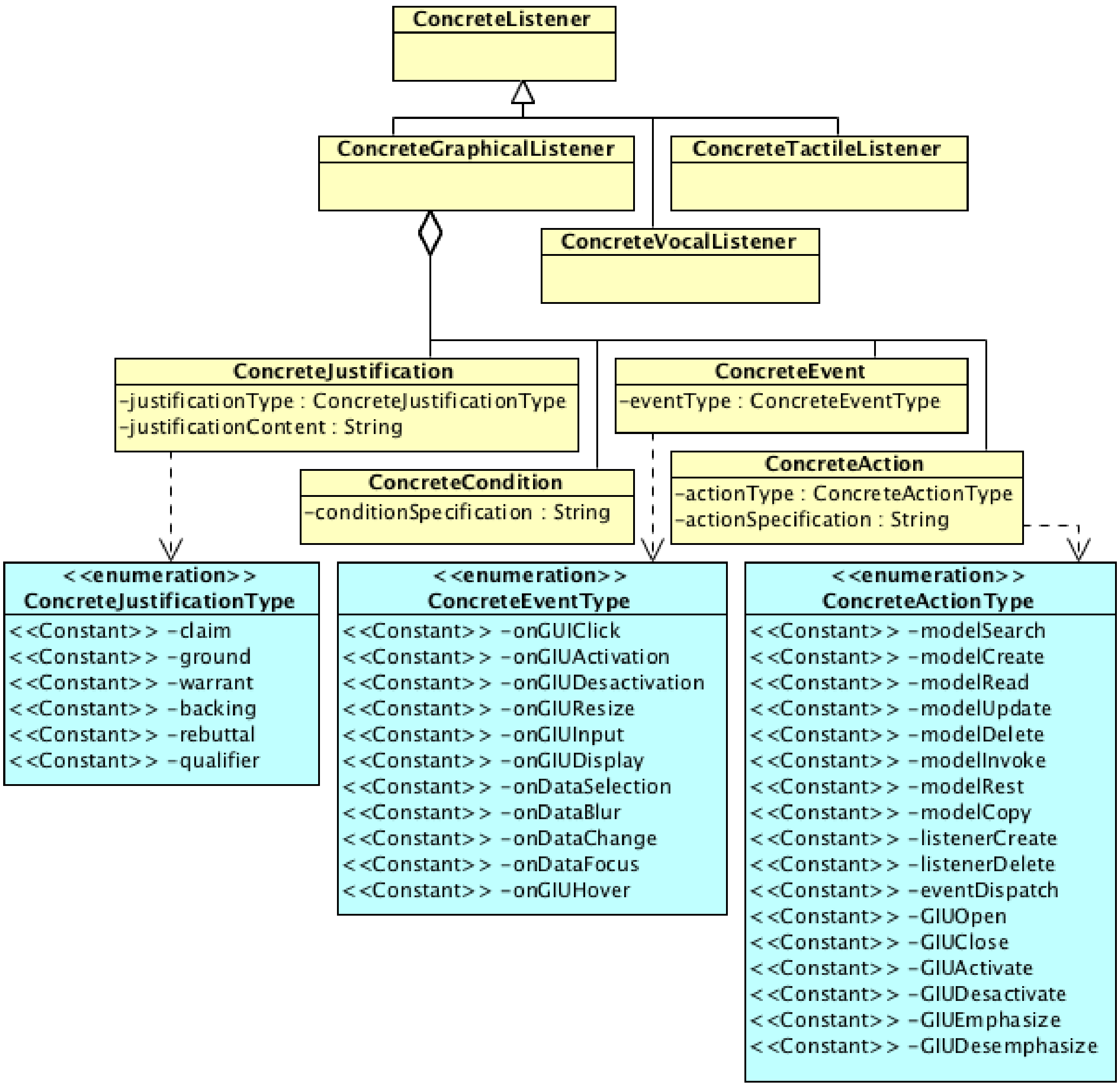
➢ *Font:* Entity describing font text.
  - *Attributes*
    - type **(FontType)**: type of font, described in *FontType* description.
    - size **(Integer)**: size of the font.
    - isBold **(Boolean)**: specifies if the font is bold.
    - isItalic **(Boolean)**: specifies if the font is italic.

➢ *FontType:*
  - *Attibutes*
    - Arial
    - Times New Roman

➢ *Color:* Entity representing a RGB color.
  - *Attributes*
    - red **(Integer)**: red component of the color (between 0 and 255)
    - green **(Integer)**: green component of the color (between 0 and 255)
    - blue **(Integer)**: blue component of the color (between 0 and 255)
    - alpha **(Float)**: transparence of the color (0 = translucent, 1 = opaque)

➢ *AligmentType*
  - *Attributes*
    - topLeft
    - topCenter
    - topcRight
    - middleLeft
    - middleCenter
    - middleRight
    - bottomLeft
    - bottomCenter
    - bottomRight

3.6.5. ConcreteListener


ConcreteListener
ConcreteGraphicalListener
ConcreteTactileListener
ConcreteVocalListener
ConcreteJustification
-justificationType : ConcreteJustificationType
-justificationContent : String
ConcreteEvent
-eventType : ConcreteEventType
ConcreteCondition
-conditionSpecification : String
ConcreteAction
-actionType : ConcreteActionType
-actionSpecification : String
<<enumeration>> ConcreteJustificationType
<<Constant>> -claim
<<Constant>> -ground
<<Constant>> -warrant
<<Constant>> -backing
<<Constant>> -rebuttal
<<Constant>> -qualifier
<<enumeration>> ConcreteEventType
<<Constant>> -onGUIClick
<<Constant>> -onGIUActivation
<<Constant>> -onGIUDesactivation
<<Constant>> -onGIUResize
<<Constant>> -onGIUInput
<<Constant>> -onGIUDisplay
<<Constant>> -onDataSelection
<<Constant>> -onDataBlur
<<Constant>> -onDataChange
<<Constant>> -onDataFocus
<<Constant>> -onGIUHover
<<enumeration>> ConcreteActionType
<<Constant>> -modelSearch
<<Constant>> -modelCreate
<<Constant>> -modelRead
<<Constant>> -modelUpdate
<<Constant>> -modelDelete
<<Constant>> -modelInvoke
<<Constant>> -modelRest
<<Constant>> -modelCopy
<<Constant>> -listenerCreate
<<Constant>> -listenerDelete
<<Constant>> -eventDispatch
<<Constant>> -GIUOpen
<<Constant>> -GIUClose
<<Constant>> -GIUActivate
<<Constant>> -GIUDesactivate
<<Constant>> -GIUEmphasize
<<Constant>> -GIUDesemphasize

- *ConcreteListener:* Allows specifying the behavior of the *AbstractInteractionUnits*. For the moment, it is refined in three simple modalities (graphical, vocal, tactile) and only the graphical modality is refined (see *ConcreteGraphicalListener* description).
- *ConcreteGraphicalListener:* Entity used to describe the behavior of a *ConcreteGraphicalIU* by using Event-Conditio-Action (ECA) rules.
- *ConcreteJustification:* The justification is a kind of motivation for the ECA rules, it is not used for describing the interface itself but more for documentation purposes.
  - *Attributes*
    - justificationType **(ConcreteJustificationType)**: type of justification, the different possibilities are described in the *ConcreteJustificationType* description.
    - justificationContent **(String)**: text representing the justification itself.

- *ConcreteJustificationType*
  - *Attributes*
    - claim: assertion put forward publicly for general acceptance with the implication that there are underlying 'reasons' that could show it to be 'well founded' and therefore entitled to be generally accepted.
      - Example: "Our organization should cut costs next quarter"
    - ground: the term 'ground' refers to the specific facts relied on to support a given claim.
      - Example: "Our organization has lost money the last 3 quarters"
    - warrant: assertion that entitles you to interpret or link the grounds (facts) as support of the claim.
      - Example: "When losing money, organizations should cut expenses as much as possible"
    - backing: the 'backing' consists of a very general set of background assumptions which, in effect, legitimize the basis for believing in the warrant. That is, if the warrant is not accepted on its surface, then the backing is called into play to add deeper support to the argument.
      - Example: "A business can't continue to lose money and stay in business"
    - rebuttal: the 'rebuttal' presents the possible exceptions or objections as to why the claim, the grounds, the warrants, or the backing may not hold for the situation under discussion.
      - Example: "That may be true in general, but not with the customers of our company. Besides that, times have changed; the economy as changed; the dollar has fallen in value.
    - qualifier: word that indicates how strongly the claim is being asserted, or how likely that something might occur.
      - Example: "probably", "certainly", "very likely", etc.
- *ConcreteEvent:* specifies the signal that triggers the invocation of the rule.
  - *Attributes*
    - eventType **(ConcreteEventType)**: type of event, the different possibilities are described in the *ConcreteEventType* description.
- *ConcreteEventType:*
  - *Attributes*
    - onGIUClick: a *ConcreteGraphicalIU* has been clicked
    - onGIUActivation: a *ConcreteGraphicalIU* has been activated
    - onGIUDesactivation: a *ConcreteGraphicalIU* has been desactivated
    - onGIUResize: a *ConcreteGraphicalIU* has been resized
    - onGIUInput: data has been entered in a *ConcreteGraphicalIU*
    - onGIUDisplay: a *ConcreteGraphicalIU* has been displayed
    - onDataSelection: a data entry element has been selected
    - onDataBlur: a data entry element has lost focus
    - onDataChange: a data entry element has lost focus after its content has been modified
    - onDataFocus: a data entry element has got focus
    - onGIUHover: a mouse pointer has been moved over an element
- *ConcreteCondition*: logical test that, if satisfied or evaluates to true, causes the action to be carried out.
  - *Attributes*
    - conditionSpecification **(String)**: logical formula representing the condition to evaluate, for the moment under the form of a String.

- *ConcreteAction:* consists of updates or invocations on the Domain Model data, or modifications of the concrete entities themselves.
  - *Attributes*
    - actionType **(ConcreteActionType)**: type of action, the different possibilities are described in the *ConcreteActionType* description.
    - actionSpecification **(String)**: kind of argument that, used in conjunction with type, allows to specify the action to launch.
      - Example: for an *ConcreteActionType* 'modelUpdate', the actionSpecification could specify which field to change and what is the new value.
- *ConcreteActionType*
  - *Attributes*
    - modelSearch: search for model elements based on logical formula
    - modelCreate: create a new model in the Domain Model
    - modelRead: read a specified field in the Domain Model
    - modelUpdate: update a field in the Domain Model
    - modelDelete: delete a field in the Domain Model
    - modelInvoke: perform a query external to the current Domain Model
    - modelReset: reset the Domain Model with the initial parameters
    - modelCopy: copy the Domain Model
    - listenerCreate: create a new listener on a specified *ConcreteGraphicalIU*
    - listenerDelete: delete a listener of a specified *ConcreteGraphicalIU*
    - eventDispatch: dispatch the event to another *ConcreteGraphicalIU*
    - GIUOpen: open a specified *ConcreteGraphicalIU*
    - GIUClose: close a specified *ConcreteGraphicalIU*
    - GIUActivate: activate a specified *ConcreteGraphicalIU*
    - GIUDesactivate: desactivate a specified *ConcreteGraphicalIU*
    - GIUEmphasize: focus in a specified *ConcreteGraphicalIU*
    - GIUDesemphasize: focus out a specified *ConcreteGraphicalIU*

3.7. Transformation

3.7.1. Overview

According to Kleppe et al. [KWB03] a *transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.* Mens et al. [MCG04] have generalized this definition saying that a transformation is also possible with multiple source models and/or multiple target models.

This section presents the UsiXML transformation meta-model. This meta-model has been defined according to the previous definitions. The aim of this meta-model is to define how transformation definitions (named transformation models in this document) are composed in UsiXML.

Transformation models are aggregations of transformation units which, in turn, are aggregations of sub-transformation units and transformation rules. Transformation units define an execution order for their sub-transformation units and transformation rules. Transformation rules have a core definition and have several rule representations, for instance ATL and/or graphs. The core definition of a transformation rule is independent of a specific transformation technology and is not executable per se. The different rule representations are related to specific transformation technologies and are executable in the corresponding compilers.

UsiXML proposes a set of conceptual models that represent diverse aspects of user interfaces. Furthermore, models are located at different abstraction levels. When following a forward engineering process, lower level models can be obtained from higher level models (reification) by means of model-to-model transformations and the user interface code can be reached through a model-to-code transformation (reification). When following a reverse engineering process, the lowest level model can be obtained from code by means of a code-to-model transformation (abstraction), and then, higher level models can be derived from lower level models by means of model-to-model transformations (abstraction). It is also possible to have model transformations at a same abstraction level, for instance when a model is being improved (reflection), or when a model is being customized for a different context of use (translation). Hence, transformation rules and transformation models can be classified as reifications, abstractions, reflections, or translations.

Furthermore, the transformation meta-model allows a transformation rule repository to be created and maintained. In the repository, transformation rules can be represented in different languages or notations. Transformation rules can be reused in different transformation units which, in turn, can be reused in different transformation models.

The transformation meta-model presented in this section allows transformation units and transformation rules to be related to design options expressed according to the QOC notation [MYBM96, LP07, LPB+07]. These design options represent design alternatives that can be evaluated according to usability criteria. The importance of the usability dimension has been highlighted in works like [SCF05]. The evaluations can be used to decide which alternative (transformation unit or rule) to execute.

3.7.2. Structure of packages

The UsiXML Transformation meta-model has been defined in a package named Transformation. However, classes of the Transformation package have relations with classes of other UsiXML packages or with classes of other packages that are defined independently of UsiXML. So, in first place, a description of the structure of packages is given.

Figure 8 presents the first nesting level. There are two packages: UsiXML and QOC. The UsiXML package contains other UsiXML related classes and packages. The QOC package contains a meta-model for the Questions-Options-Criteria (QOC) notation [MYBM96] which is useful for analyzing the rational of design decisions.

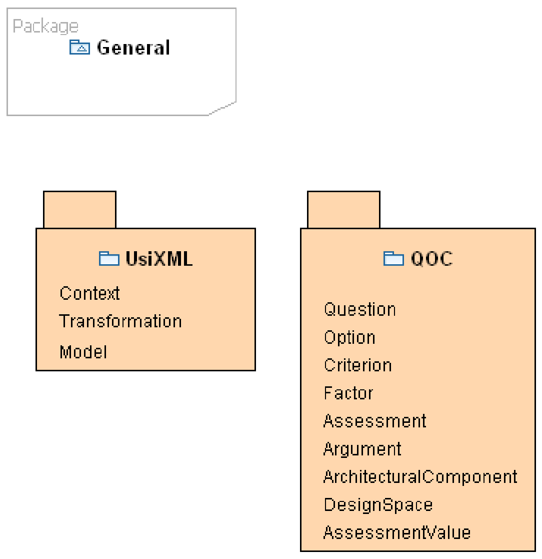


Figure 8. Packages related to the Transformation meta-model

Figure 9 illustrates the contents of the UsiXML package. There are two other sub-packages: Context and Transformation. The Context package defines the UsiXML Context meta-model (described in Section 5.3 and the Transformation package defines the UsiXML Transformation meta-model. These are the two relevant UsiXML packages when describing the UsiXML Transformation meta-model. However, the UsiXML package contains more sub-packages related to the other meta-models of UsiXML (for instance, Abstract User Interface, Concrete User Interface, etc.). The abstract class Model represents any UsiXML model and it is specialized in the different sub-packages. For instance, in the Transformation package we have a class named TransformationModel which is a specialization of Model.

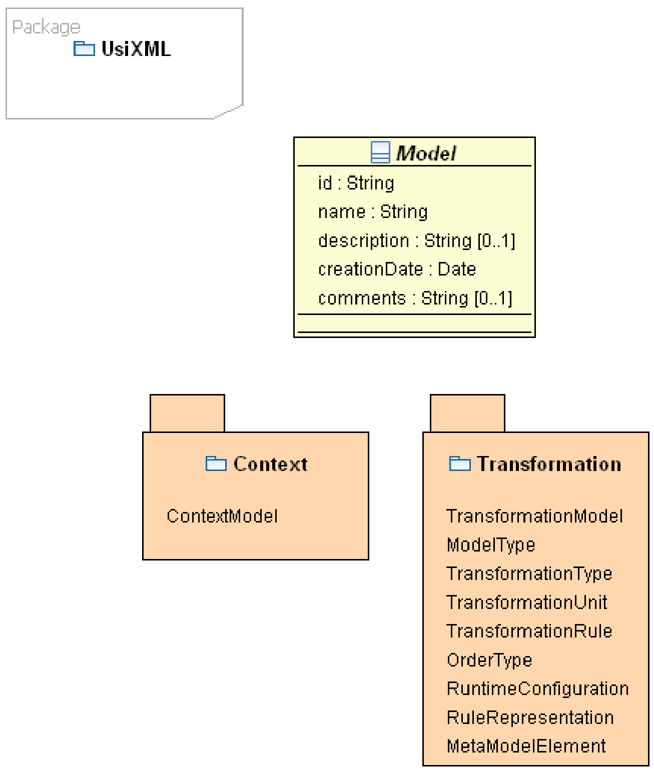


Figure 9. Contents of the UsiXML package

Since classes of the Transformation package have relations with classes of the packages Context and QOC, the contents of these packages are presented first, and the contents of the Transformation package are presented later.

3.7.3. Package Context

This package contains the UsiXML Context meta-model that aims to describe the context of use (i.e., user, platform, environment) in which the generated interface will be executed. This package is defined in Section 5.3.

3.7.4. Package QOC

This package supports design rationale based on the QOC (Questions, Options, Criteria) notation [MYBM96]. It supports the exploration of options during design processes. The analysis starts from a set of design questions to evaluate different options according to different criteria.

This package has been defined according to [LP07] and [LPB+07].

Figure 10 illustrates the QOC meta-model.

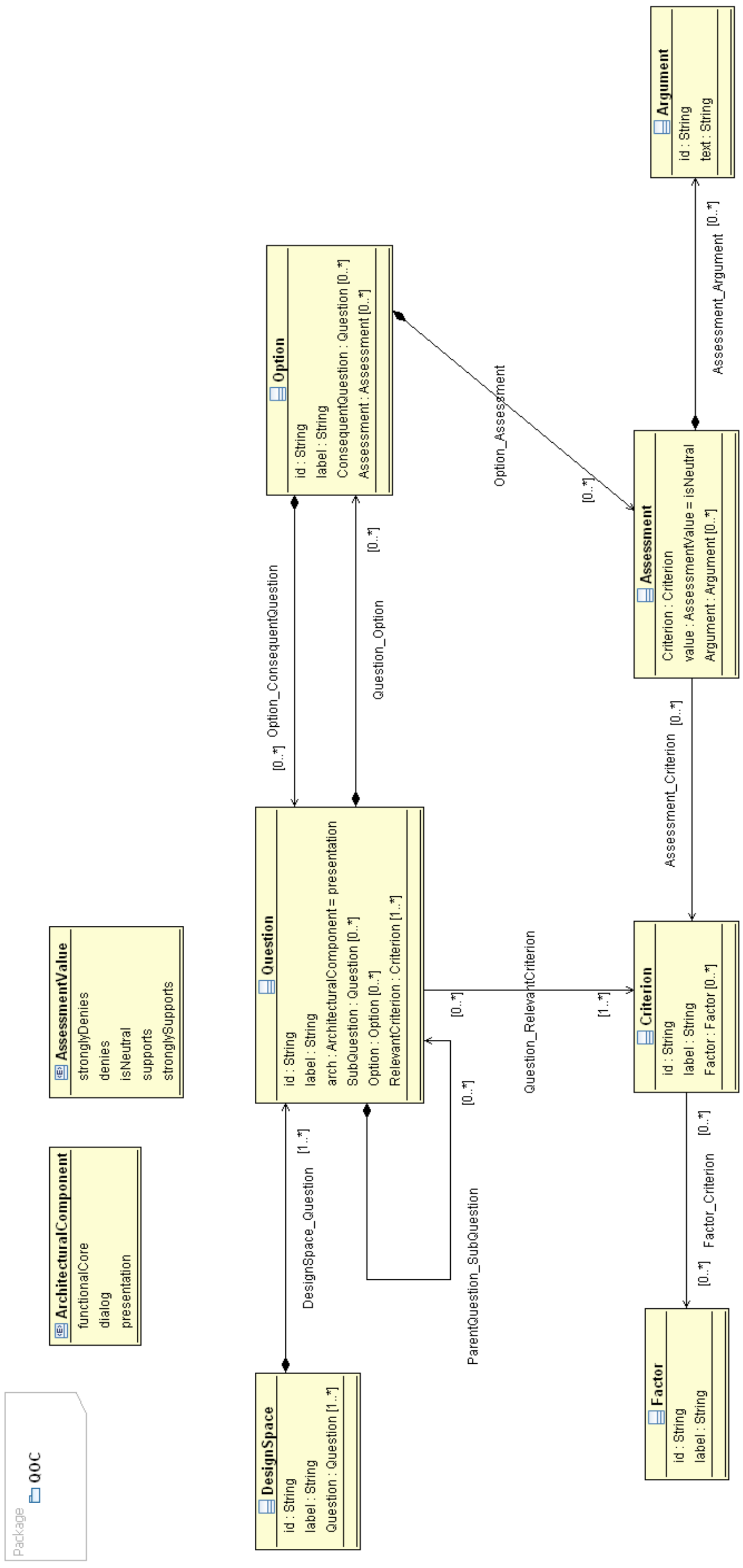


Figure 10. QOC meta-model

A *design space* allows the design of an artifact to be analyzed taking into account the identification, comparison, and assessments of alternative designs for the artifact. Hence, the design space is composed of *questions* which represent key design issues to be analyzed. General questions can be decomposed into more specific sub-questions. Furthermore, questions gather a set of *options*, each of which represents an alternative design or solution. Complex options can be decomposed again in consequent questions. Each question is related to a set of relevant *criteria* in order to assess each option of the question against each of the relevant criteria. Each of these *assessments* holds a value that indicates if, in the context of the question, the option supports, is neutral, or denies the criterion. Furthermore, each assessment has a set of *argumentations* that explain the valuation made. Criteria can be related to more general *factors*.

- *DesignSpace:* represents the analysis of the design of an artifact. The analysis is intended to identify, compare, and assess different options for the design of an artifact.
  - *Attributes:*

    - id **(String, required)**: unique identifier.

    - label **(String, required)**: name of the design space or short description.
  - *Relationships:*
    - A design space is composed of one or many questions.
  - *Examples:*
    - A design space could be defined to analyze the design of an Automated Teller Machine (ATM). See [MYBM96] for the complete example.
    - A design space could be defined to analyze how to transform from an Abstract User Interface model to a Concrete User Interface model.
- *Question:* represents a key design issue.
  - *Attributes:*

    - id **(String, required)**: unique identifier.

    - label **(String, required)**: text of the question.

    - arch **(ArchitecturalComponent, required)**: defines a relationship between the question and a component of the software architecture.
      - Possible values: defined by the ArchitecturalComponent enumeration (functionalCore, dialog, and presentation).
      - Default value: presentation.
  - *Relationships:*
    - A question belongs to one design space.
    - A question can be composed of cero or many sub-questions.
    - A sub-question belongs to one parent question.
    - A question can be composed of cero or many options.
    - A question has one or many relevant criterion.
    - A consequent question belongs to one option.
  - *Restrictions:*
    - If a question does not have options, it should have at least one sub-question.
    - If a question does not have sub-questions, it should have at least two options.
  - *Examples:*
    - In the design space that analyzes how to transform an Abstract User Interface model into a Concrete User Interface model, we can have the following question: what concrete graphical interactor should be used as target of abstract selection interactors? This question is related to the presentation component of the software architecture.
- *Option:* represents a possible answer to a question or, in other words, represents a design solution.

- *Attributes:*

  - id **(String, required)**: Option's identifier.

  - label **(String, required)**: Option's name or short description.

- *Relationships:*
  - An option belongs to one question.
  - An option can be composed of cero or many consequent questions.
  - An option can be composed of cero or many assessments.
- *Restrictions:*
  - The number of assessments of an option is equal to the number of relevant criterion of the question of the option. An option must have one assessment for each relevant criterion of the question of the option.
- *Examples:*
  - Combo box and radio box are two concrete options for transforming abstract selection interactors.

➢ *Criterion:* allows assessing and comparing options. It is intended to perform qualitative comparisons between options.

- *Attributes:*

  - id **(String, required)**: Criterion's identifier.

  - label **(String, required)**: Criterion's name or short description.

- *Relationships:*
  - A criterion can be related to cero or many factors.
  - A criterion can be relevant to cero or many questions.
  - A criterion can participate in cero or many assessments.
- *Examples:*
  - Information density and brevity are criteria that can be associated as relevant with regard to the question of what concrete graphical interactor should be used as target of abstract selection interactors. The relevant criteria will be used to compare the options for transforming abstract selection interactors. Information density refers to the amount of information that is displayed. It should not be too large. Brevity refers to the amount of actions that the user must perform in order to achieve a goal. This amount should not be too large.

➢ *Factor:* allows requirements expressed by the clients and/or users to be represented. Factors correspond to high-level requirements such as learnability, safety, usability, etc. [LP07].

- *Attributes:*

  - id **(String, required)**: Factor identifier.

  - label **(String, required)**: Factor name or short description.

- *Relationships:*
  - A factor can be related to cero or many criteria.
- *Examples:*
  - Usability is a factor that can be related to the two previously mentioned criteria: information density and brevity.

➢ *Assessment:* allows options of a question to be evaluated against each of the relevant criteria of the question. The evaluation indicates if, in the context of the question, the option supports, denies, or is neutral with respect to each relevant criterion.

- *Attributes:*

  - value **(AssessmentValue)**: evaluates an option with respect to a relevant criteria. Allows a qualitative and comparative analysis amongst the various options related to a question.

    - Possible values: defined by the AssessmentValue enumeration (strongly denies, denies, is neutral, supports, strongly supports).
    - Default value: is neutral.

- *Relationships:*
  - An assessment belongs to one option.
  - An assessment is made in relation to one criterion.
  - An assessment can be composed of cero or many arguments.
- *Restrictions:*
  - There should be an assessment for each pair (option, relevant criterion) of a question.
- *Examples:*
  - This class allows us to express that combo box is an option that supports information density but denies brevity. On the contrary, radio box denies information density and support brevity.

➢ *Argument:* allows the assessments of options against criteria to be explained.

- *Attributes:*

  - id **(String, required)**: Argument identifier.

  - text **(String, required)**: Text that exposes the argument.

- *Relationships:*
  - An argument belongs to one assessment.
- *Examples:*
  - Combo box supports information density because of the following argument:
    - Combo boxes do not display all of them items at a time.
  - Combo box denies brevity because of the following argument:
    - The user has to make clicks or move a slider to see all possible items.
  - Radio box supports brevity because of the following argument:
    - The user does not have to make any actions to see all the possible items.
  - Radio box denies information density because of the following argument:
    - All possible items are displayed at the same time.

[LP07] and [LPB+07] can be consulted for more information about QOC.

3.7.5. Package Transformation

This section presents the classes that compose the package Transformation. Figure 11 shows the classes of this package. This package defines the UsiXML Transformation meta-model.

3.7.5.1. Transformation Rules and Rule Representations

A *transformation rule* is intended to specify how one or more elements of one or more source model types (source meta-model elements) are transformed into one or more elements of one or more target model types (target meta-model elements). A transformation rule can also specify how one or more elements of a set of model types are transformed into one or more elements of the same set of model types in order to incorporate enhancements or customizations for different contexts of use.

Taking into account the sets of source and target meta-model elements and the source and target contexts of use, a transformation rule can be categorized as a:

- **Reification**: in this case, the abstraction level of the source meta-model elements is higher than the abstraction level of the target meta-model elements. The source and target contexts of use are the same.
- **Abstraction**: in this case, the abstraction level of the source meta-model elements is lower than the abstraction level of the target meta-model elements. The source and target contexts of use are the same.

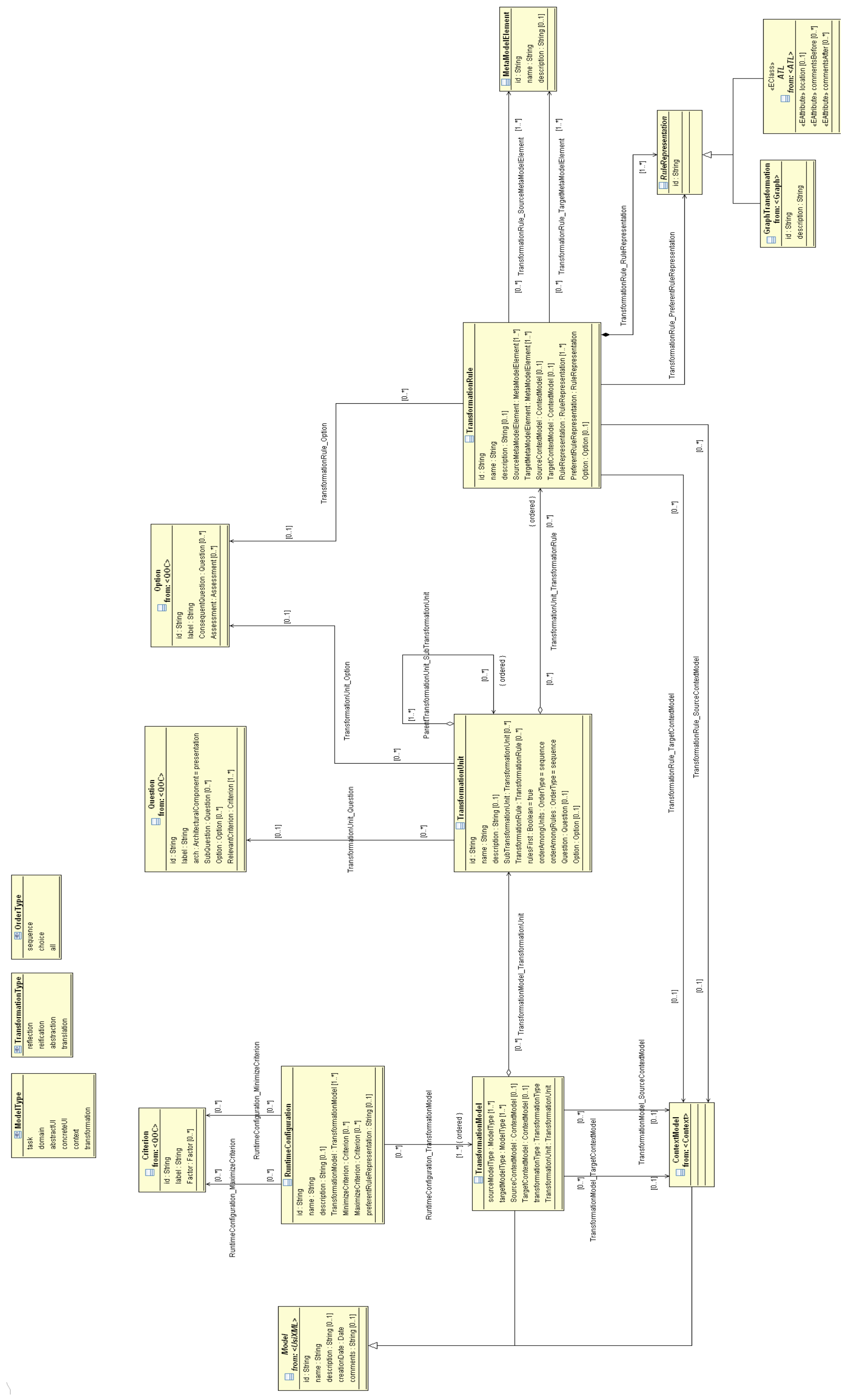


Figure 11. Transformation meta-model

- **Reflection**: in this case, the abstraction level of the source meta-model elements is equal to the abstraction level of the target meta-model elements (this is, the set of source model types is the same than the set of target model types). The source and target contexts of use are the same.
- **Translation**: in this case, the abstraction level of the source meta-model elements is equal to the abstraction level of the target meta-model elements (this is, the set of source model types is the same than the set of target model types). The source and target contexts of use are different.

The *core representation* of a transformation rule only specifies which are the source and target meta-model elements and the source and target contexts of use. This core representation of a transformation rule, with this information, is independent of any specific transformation technology and is not executable per se. However, a transformation rule is composed of one or more *rule representations*, for instance ATL and/or graphs (other types of rule representations, for instance QVT, could be added if necessary). These rule representations are related to specific transformation technologies and are executable in the corresponding compilers. Each rule representation is restricted to use only the source and target meta-model elements and the source and target contexts of use defined in the core representation. Furthermore, each transformation rule has a designated preferred rule representation that will have priority over other rule representations for execution.

- *TransformationRule:* A Transformation Rule is a unitary transformation operation. It is a description of how one or more constructs in the source models can be transformed into one or more constructs in the target models. This element is a core representation for transformation rules.
  - *Attributes:*

    - id **(String, required)**: Transformation rule's identifier.

    - name **(String, required)**: Transformation rule's name.

    - description **(String, optional)**: Transformation rule's goal.

    - transformationType **(TransformationType, required, derived)**: category or type of the transformation rule.

      - Possible values: defined by the TransformationType enumeration (reflection, reification, abstraction, translation).
      - Derivation rules:
        - Reflection: the abstraction level of the source meta-model elements is equal to the abstraction level of the target meta-model elements (this is, the set of source model types is the same than the set of target model types) and the source context model is the same than the target context model.
        - Reification: the set of source meta-model elements corresponds to a higher abstraction level than the set of target meta-model elements. The source context model is the same than the target context model.
        - Abstraction: the set of source meta-model elements corresponds to a lower abstraction level than the set of target meta-model elements. The source context model is the same than the target context model.
        - Translation: the abstraction level of the source meta-model elements is equal to the abstraction level of the target meta-model elements (this is, the set of source model types is the same than the set of target model types). The source context model is different than the target context model.

- *Relationships:*
  - A transformation rule has one or many source meta-model elements.
  - A transformation rule has one or many target meta-model elements.
  - A transformation rule can have cero or one source context model.
  - A transformation rule can have cero or one target context model.
  - A transformation rule is composed of one or more rule representations.
  - A transformation rule has one preferred rule representation.
  - A transformation rule can be aggregated in cero or many transformation units.
  - A transformation rule can be linked to cero or one option.
- *Restrictions:*
  - If there is a source context model there should be a target context model and vice versa.
  - If the set of source meta-model elements is different than the set of target meta-model elements, then the source context of use must be the same than the target context of use.
  - If there are source and target context models and they are different, then the set of source meta-model elements must be equal to the set of target meta-model elements.

➢ *RuleRepresentation:* is an abstract class that represents a transformation rule in a specific transformation language that can be executed. Each of these rule representations is governed by a specific notation and a corresponding compiler is able to execute the rule. For instance, a transformation rule can have two different rule representations, one in ATL and another one with graphs. Packages Graph and ATL define these types of rule representations (see Section 3.7.6).

- *Attributes:*

  - id **(String, required)**: Rule representation identifier.
- *Relationships:*
  - A rule representation belongs to one transformation rule.
  - A rule representation is the preferred one just in one transformation rule.
- *Restrictions:*
  - Each rule representation is restricted to use only the source and target meta-model elements and the source and target contexts of use defined in the core representation of the rule.

3.7.5.2. Transformation Units

A *transformation unit* gathers transformation rules and sub-transformation units and defines an execution order for them.

Furthermore, a transformation unit can be linked to a question of the QOC package. In this case, the transformation unit implements the design issue represented by the question. If the transformation unit linked to a question gathers a set of transformation rules, then each of these transformation rules must match an option of the question, this is, each transformation rule implements its corresponding option. Figure 12 illustrates this case. Transformation unit TU1 has two transformation rules TR1 and TR2. TU1 is associated to (implements) question Q1. Question Q1 has two options, O1 and O2. TR1 is associated to (implements) O1 and TR2 is associated to (implements) O2.

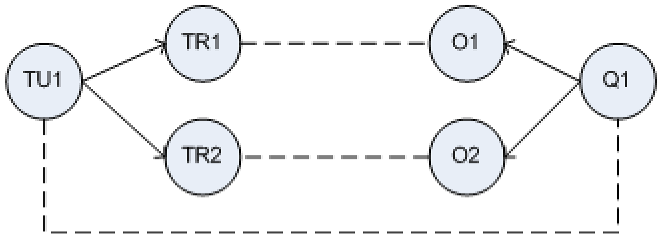


Figure 12. Transformation unit related to a question and transformation rules related to options

When an option of a question is complex, just one transformation rule can be not enough to implement the option. In this case, a sub-transformation unit can be used to implement an option. Figure 13 illustrates this case. O1 is implemented by TU11 and O2 is implemented by TU12. In turn, TU11 and TU12 can have other sub-transformation units and transformation rules.

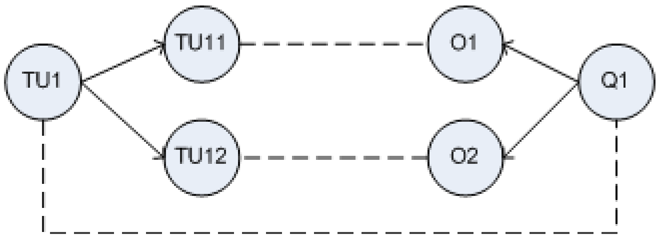


Figure 13. Transformation unit related to a question and sub-transformation units related to options

With regard to execution orders, a transformation unit can specify that its set of transformation rules will be executed first, followed by the execution of its set of sub-transformation units, or vice versa.

Furthermore, there are three options for specifying how to execute the transformation rules of a transformation unit:

- **Sequence**: all transformation rules are executed sequentially.
- **All**: all transformation rules are executed in random order.
- **Choice**: only one transformation rule is executed. This option can only be used when the transformation unit is linked to a question and each transformation rule is linked to an option of the question. The transformation rule to be executed will be the one that is linked to the option that best supports a set of criteria to be maximized and a set of criteria to be minimized during execution (see the class *RuntimeConfiguration*). The assessments of options with regard to criteria are consulted in order to identify this transformation rule. See Section 3.7.5.4 for an example of the selection of the transformation rule to be executed.

The same three options are used for specifying how to execute the sub-transformation units of a transformation unit:

- **Sequence**: all sub-transformation units are executed sequentially.
- **All**: all sub-transformation units are executed in random order.
- **Choice**: only one sub-transformation unit is executed. This option can only be used when the transformation unit is linked to a question and each sub-transformation unit is linked to an option of the question. The sub-transformation unit to be executed will be the one that is linked to the option that best supports a set of criteria to be maximized and a set of criteria to be minimized during execution (see the class *RuntimeConfiguration*). The

assessments of options with regard to criteria are consulted in order to identify this sub-transformation unit.

- *TransformationUnit:* gathers a set of sub-transformation units and a set of transformation rules and its purpose is to specify the order in which sub-transformation units and transformation rules will be executed.
  - *Attributes:*

    - id **(String, required)**: transformation unit identifier.

    - name **(String, required)**: name of the transformation unit.

    - description **(String, optional)**: description of the transformation unit.

    - rulesFirst **(Boolean, required)**: each transformation unit gathers a set of transformation rules and, apart from that, it can gather several sub-transformation units. It is important to establish the execution order among the set of transformation rules and the set of sub-transformation units. If the attribute rulesFirst is true, the set of transformation rules will be executed before executing the set of sub-transformation units. Otherwise, the set of sub-transformation units will be executed first.

      - Default value: true.

    - orderAmongUnits **(OrderType, required)**: a transformation unit can gather other sub-transformation units. The aim of this attribute is to specify how these sub-transformation units will be executed.

      - Possible values: defined by the OrderType enumeration (sequence, choice, all).
        - Sequence: all sub-transformation units will be executed sequentially.
        - All: all sub-transformation units will be executed randomly.
        - Choice: only one sub-transformation unit will be executed. The decision of which sub-transformation unit will be executed is based on QOC, as previously explained.
      - Default value: sequence.

    - orderAmongRules **(OrderType, required)**: a transformation unit can gather transformation rules. The aim of this attribute is to specify how these transformation rules will be executed.

      - Possible values: defined by the OrderType enumeration (sequence, choice, all).
        - Sequence: all transformation rules will be executed sequentially.
        - All: all transformation rules will be executed randomly.
        - Choice: only one transformation rule will be executed. The decision of which transformation rule will be executed is based on QOC, as previously explained.
      - Default value: sequence.
  - *Relationships:*
    - A transformation unit aggregates an ordered set of sub-transformation units.
    - A transformation unit aggregates an ordered set of transformation rules.
    - A sub-transformation unit is aggregated in one or more transformation units.
    - A transformation unit can be linked to (implement) cero or one question.
    - A transformation unit can be linked to (implement) cero or one option.
    - A transformation unit can be aggregated in cero or many transformation models.
  - *Restrictions:*
    - A transformation unit must aggregate at least one transformation rule or sub-transformation unit.
    - Transformation rules are ordered in transformation units.
    - Sub-transformation units are ordered in transformation units.

- A transformation unit cannot be linked to a question and to an option.
- If a transformation unit is linked to a question, then it should aggregate only transformation rules or only sub-transformation units, not a mix of them.
- If a transformation unit linked to a question aggregates transformation rules,
  - The attribute orderAmongRules must have value choice.
  - There should be exactly one transformation rule for each option of the question.
- If a transformation unit linked to a question aggregates sub-transformation units,
  - The attribute orderAmongUnits must have value choice.
  - There should be exactly one sub-transformation unit for each option of the question.
- If a transformation unit is linked to an option, then it should be a sub-transformation unit of another transformation unit that is linked to the question of the option.

- *Examples:*

<u>*Example for order of executions*</u>

Suppose we have a transformation unit, TU1, with transformation rules, TR1 and TR2, and with sub-transformation units, TU2 and TU3. Figure 14 illustrates the case.

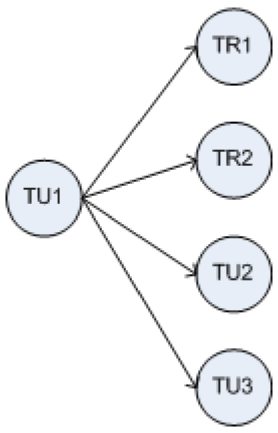


Figure 14. A transformation unit with transformation rules and sub-transformation units

If the value of the attribute rulesFirst of TU1 is:

- “True”, then the set of transformation rules will be executed before the set of sub-transformation units.
- “False”, then the set of sub-transformation units will be executed before the set of transformation rules.
- Once the order between the sets of transformation rules and sub-transformation units of TU1 has been defined, it is necessary to specify the order of execution inside the sets.
  - The order of execution of TR1 and TR2 depends on the value of the attribute orderAmongRules of TU1. If the value is:
    - “Sequence”, then TR1 and TR2 will be executed in sequential order.
    - “All”, then TR1 and TR2 will both be executed, in random order.
    - “Choice”, then just TR1 or just TR2 will be executed. Another example illustrates the selection process (see Section 3.7.5.4).

- The order of execution of TU2 and TU3 depends on the value of the attribute orderAmongUnits of TU1. If the value is:
    - “Sequence”, then TU2 and TU3 will be executed in sequential order.
    - “All”, then TU2 and TU3 will both be executed, in random order.
    - “Choice”, then just TU2 or just TU3 will be executed. Another example illustrates the selection process (see Section 3.7.5.4).

3.7.5.3. Transformation Model

A *transformation model* gathers a set of transformation rules (organized in transformation units) that together describe how models of one or more source model types are transformed into models of one or more target model types. A transformation model can also be used to specify how to improve models or how to customize them for different contexts of use.

Taking into account the sets of source and target model types and the source and target contexts of use, a transformation model, as well as a transformation rule, can be categorized as a:

- **Reification**: in this case, the abstraction level of the source model types is higher than the abstraction level of the target model types. The source and target contexts of use are the same.
- **Abstraction**: in this case, the abstraction level of the source model types is lower than the abstraction level of the target model types. The source and target contexts of use are the same.
- **Reflection**: in this case, the abstraction level of the source model types is equal to the abstraction level of the target model types (this is, the set of source model types is the same than the set of target model types). The source and target contexts of use are the same.
- **Translation**: in this case, the abstraction level of the source model types is equal to the abstraction level of the target model types (this is, the set of source model types is the same than the set of target model types). The source and target contexts of use are different.

Furthermore, there must be a correspondence between a transformation model and its transformation rules. This is:

- The set of source meta-model elements that participates in transformation rules of a transformation model must correspond to the set of source model types of the transformation model.
- The set of target meta-model elements that participates in transformation rules of a transformation model must correspond to the set of target model types of the transformation model.
- The source context of use used in transformation rules of a transformation model must be the same than the source context of use of the transformation model.
- The target context of use used in transformation rules of a transformation model must be the same than the target context of use of the transformation model.

As a consequence of these correspondences:

- If the transformation model is a reification, it will contain transformation rules which are reifications.
- If the transformation model is an abstraction, it will contain transformation rules which are abstractions.

- If the transformation model is a translation, it will contain transformation rules which are translations.
- If the transformation model is a reflection, it will contain transformation rules which are reflections.

➢ *TransformationModel:* is a specialization of the class Model from the UsiXML package. Aggregates transformation units which aggregate transformation rules that specify how to transform models.

- *Attributes:*

  - sourceModelType **(ModelType, required)**: allows to specify the types of the models (or meta-model names) that will be the input of the transformation process. It is possible to specify more than one.

    - Possible values: defined by the ModelType enumeration (task, domain, abstractUI, concreteUI, context, transformation). The enumeration can be extended if needed.

  - targetModelType **(ModelType, required)**: allows to specify the types of the models (or meta-model names) that will be the output of the transformation process. It is possible to specify more than one.

    - Possible values: defined by the ModelType enumeration (task, domain, abstractUI, concreteUI, context, transformation). The enumeration can be extended if needed.

  - transformationType **(TransformationType, required, derived)**: category or type of the transformation model.

    - Possible values: defined by the TransformationType enumeration (reflection, reification, abstraction, translation).
    - Derivation rules:
      - Reflection: the set of source model types is the same than the set of target model types and the source context model is the same than the target context model.
      - Reification: the set of source model types corresponds to a higher abstraction level than the set of target model types. The source context model is the same than the target context model.
      - Abstraction: the set of source model types corresponds to a lower abstraction level than the set of target model types. The source context model is the same than the target context model.
      - Translation: the set of source model types is the same than the set of target model types. The source context model is different than the target context model.

- *Relationships:*
  - A transformation model is a specialization of Model.
  - A transformation model can have cero or one source context model.
  - A transformation model can have cero or one target context model.
  - A transformation model aggregates one transformation unit.
  - A transformation model can participate in cero or many runtime configurations.

- *Restrictions:*
  - If there is a source context model there should be a target context model and vice versa.
  - If the set of source model types is different than the set of target model types, then the source context of use must be the same than the target context of use.
  - If there are source and target context models and they are different, then the set of source model types must be equal to the set of target model types.

- A transformation model must aggregate transformation rules such that their set of source meta-model elements must be correspondent with the set of source model types of the transformation model.
- A transformation model must aggregate transformation rules such that their set of target meta-model elements must be correspondent with the set of target model types of the transformation model.
- A transformation model must aggregate transformation rules whose source context of use is the same than the source context of use of the transformation model.
- A transformation model must aggregate transformation rules whose target context of use is the same than the target context of use of the transformation model.

3.7.5.4. Runtime Configuration

A *runtime configuration* can be defined in order to organize the execution of model transformations. A runtime configuration indicates the transformation models to be executed and their order of execution.

Furthermore, the runtime configuration defines which rule representation will be used during the execution of all the involved transformation models. For instance, the runtime configuration can indicate that the ATL representation will be used for the execution of the involved transformation models. In this case, for each transformation rule that must be executed, it will be its ATL rule representation the one that will be actually executed. If a transformation rule does not have an ATL representation, then the transformation rule preferred rule representation will be used instead.

The runtime configuration also defines a set of criteria to be maximized and a set of criteria to be minimized during the execution of the involved transformation models. These criteria will be used when analyzing which transformation rule or transformation unit (associated to an option of a question) will be executed in a "choice" situation. The assessments of the option with respect to these criteria will be analyzed, and the option that best supports the minimization and maximization requirements will be chosen.

➢ *RuntimeConfiguration:* organizes the execution of transformation models. Defines the transformation models involved and their order of execution. Defines the rule representation to be used, a set of criteria to be minimized, and a set of criteria to be maximized during execution.
   - *Attributes:*

     - id **(String, required)**: runtime configuration identifier.

     - name **(String, required)**: name of the runtime configuration.

     - description **(String, optional)**: description of the runtime configuration.

     - preferredRuleRepresentation **(String, optional)**: defines the rule representation to be used during the execution of transformation models. If no preferred rule representation is defined in a runtime configuration, or if a transformation rule does not have the corresponding rule representation, then the preferred rule representation of the transformation rule will be used during the execution.
   - *Relationships:*
     - A runtime configuration has one or more transformation models.
     - A runtime configuration can have cero or many criteria to maximize.
     - A runtime configuration can have cero or many criteria to minimize.
   - *Restrictions:*
     - A criterion that is to be maximized in a runtime configuration cannot be minimized in the same runtime configuration, and vice versa.
   - *Examples:*

Example for the selection of the option to be executed

Consider the example that has been described in Section 3.7.4 for QOC.

The question is:

- Q1: what concrete graphical interactor should be used as target of abstract selection interactors?

The options are:

- O1: combo box
- O2: radio box

The criteria relevant for the question are:

- C1: information density (regarding the amount of information displayed)
- C2: brevity (regarding the amount of steps the user must perform to achieve a goal)

The assessments are:

- O1 supports C1
- O1 denies C2
- O2 denies C1
- O2 supports C1

Suppose Q1 is related to a transformation unit TU1 with transformation rules TR1 and TR2. TR1 implements O1 and TR2 implements O2. Figure 12 illustrates this case.

Suppose TU1 is defined in a transformation model TM.

Suppose we are going to execute TM and we prepare a runtime configuration RC which specifies that during execution C1 must be maximized and C2 must be minimized. In this case, during the execution, TR1 will be selected and executed because it represents the option (O1) whose valuations best support the requirements expressed in RC.

If RC would have specified that C1 must be minimized and C2 must be maximized, then TR2 would have been executed.

Al algorithm that takes as input the set of criteria to minimize and the set of criteria to maximize, that analyzes the assessments of the option-criteria pairs, and that outputs the best available option (transformation rule/ unit) must be implemented.

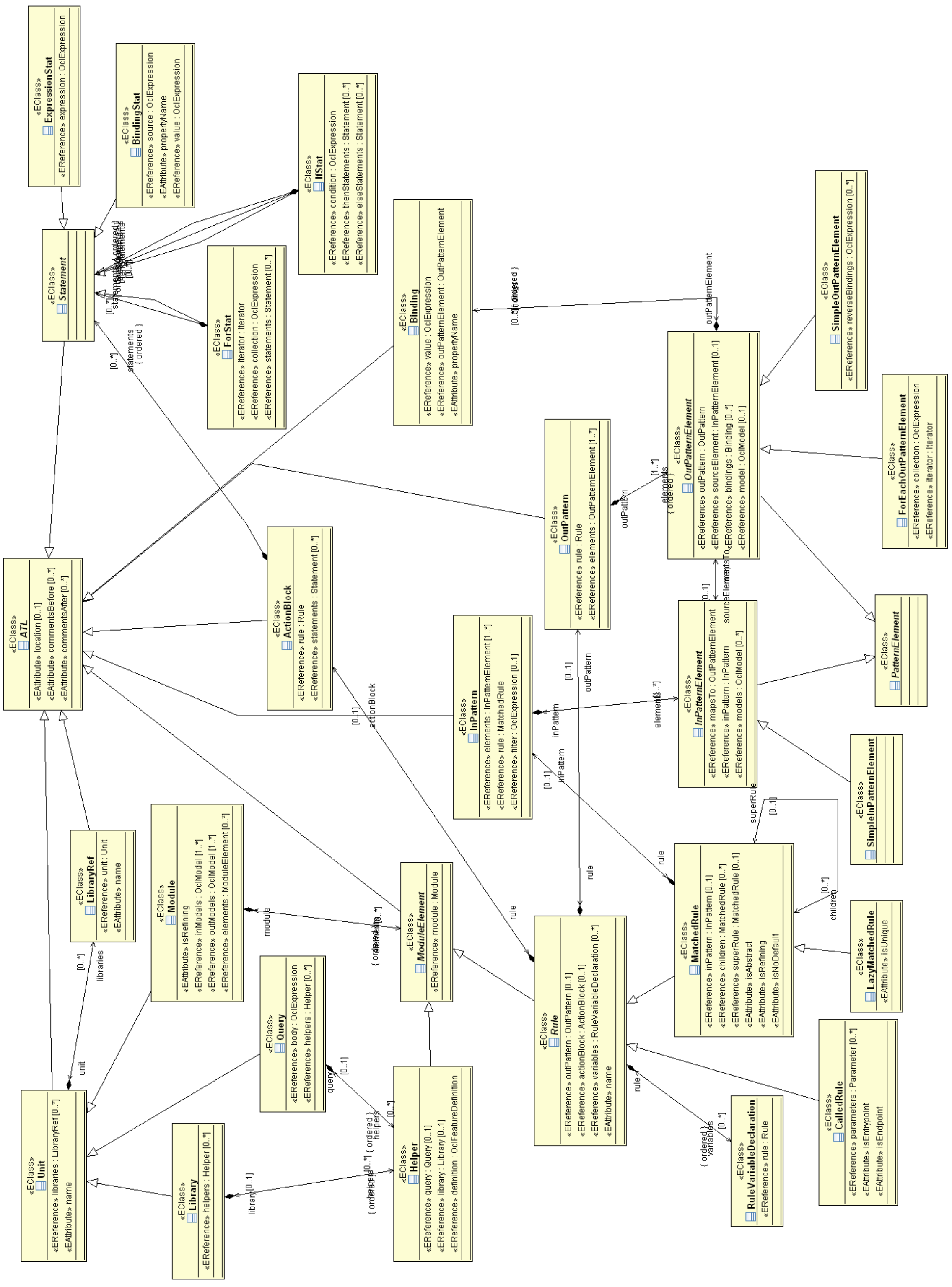
3.7.6. Connections with other transformation languages

A transformation rule has different rule representations. These rule representations specify the transformation rule according to a specific transformation technology. In this way, the notation of the specific technology is used to represent the rule, and corresponding tools and compilers can be used to execute the rule.

In this section two transformation technologies that can be used to represent transformation rules are presented: ATL and graphs. More transformation technologies could also be adopted, for instance, QVT.

3.7.6.1. ATL Rule Representations

The transformation meta-model has been linked with ATL. We have used the ATL meta-model (Figure 15) defined in the URL http://dev.eclipse.org/viewcvs/viewvc.cgi/org.eclipse.m2m/org.eclipse.atl/plugins/org.eclipse.m2m.atl.engine/src/org/eclipse/m2m/atl/engine/resources/ATL-0.2.ecore?view=log&root=Modeling_Project&pathrev=ATL_before_3_0_0. Information about the meaning of each class can be found in [Jou05] and in http://www.eclipse.org/m2m/atl/doc/. We have only



done one change in the original ATL meta-model; we have change the name of the root class (old *NamedElement)* for *ATL*, since it is more representative to be used as a specialization of the class *RuleRepresentation.*

Figure 15. ATL meta-model

3.7.6.2. Graph Rule Representations

Graph grammar transformation rules are based on the specification of how a source graph should be transformed in order to produce a target graph. The basic notion underlying graph transformation is matching some parts of the source graph and to transform them according to what the transformation rule specifies. Example of relevant general purpose graph transformation environments are AAG [T00] and GREAT [A03]. Graph transformation approach has been adapted for its use in user interface design in [LVM+04], [LMR09].

A graph transformation rule is composed of three parts: a left-hand side, a right-hand side and, optionally, a negative application condition. These three parts include meta-model elements. The meta-model elements create a graph structure. Any of the attributes of a meta-model element can include an attribute condition, which is expressed by means of an expression. Rule designers describe in the left-hand side of the rule a graph structure. The graph transformation engine will match the graph structure in the left-hand side of the rule in the source graph. The matching parts in the source graph will be transformed according to the graph structure in the right-hand side of the rule. Finally, the negative application condition is used to prevent infinite loops, since the transformation engine keeps searching for matches until no one is found. Therefore, if for instance the right-hand side of the rule does not modify the part matched with the left-hand side, and instead it adds extra elements, the transformation engine would enter into an infinite loop. Thus, a negative application condition should be used.

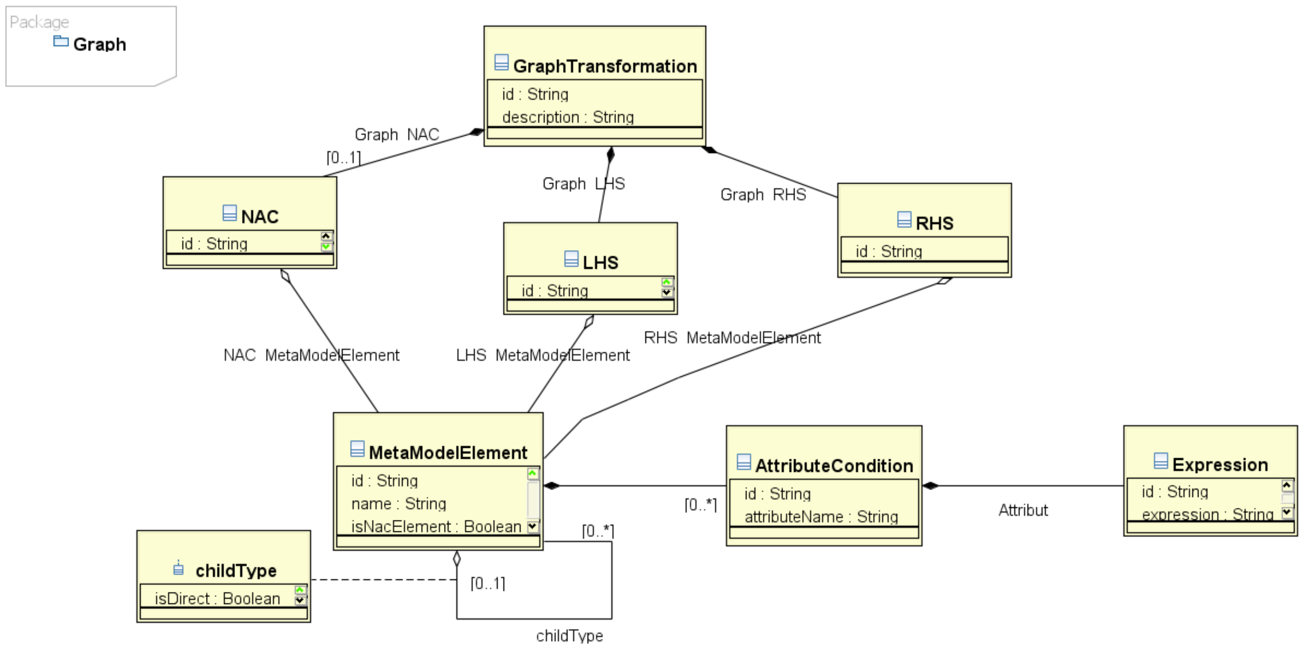


Figure 3-16. Graph transformation meta-model.

3.7.6.2.1 Graph Transformation

- *GraphTransformation:* represents a graph transformation rule.
    - *Attributes:*

      - id **(String, required)**: identifier of the graph transformation.

      - description **(String, optional)**: description of the graph transformation. It can be used to document the transformation, in the same way as a regular program is documented.
    - *Relationships:*
        - A graph transformation aggregates a LHS and a RHS elements, and optionally a NAC element.
    - *Restrictions:*
        - A graph transformation must include at least one LHS element.
        - A graph transformation must include at least one RHS element.

3.7.6.2.2 Meta Model Elements

- *MetaModelElement:* is a simplification to represent any element included in any meta-model of UsiXML.
    - *Attributes:*

      - id **(String, required)**: identifier of the meta-model element.

      - name **(String, optional)**: name of the meta-model element.

      - description **(String, optional)**: description of the meta-model element.

      - isNacElement **(String, optional)**: the attribute isNacElement is used to specify which elements included in the NAC are actually acting as NAC elements. In some approaches all the elements included in the NAC part of the rule play the role of NAC, but in some others approaches the nodes playing the role of NAC must be explicitly marked.
    - *Relationships:*
        - A meta-model element has hierarchical relationship with other meta-model elements, shaping a tree-based structure.
        - Although it is not presented in the model, for the sake of clarity, actually any relationship between two meta model elements defined in any meta model of UsiXML could also appear between the meta model elements used in a graph transformation specification.
        - The mapping relationship between meta model elements represents the mappings specified between those elements appearing as aggregates in the LHS, RHS and NAC elements of a graph transformation.
        - In the hierarchical decomposition has an associated class including the attribute *direct.* This attribute reflects whether a child is direct or not, since in some graph transformation approaches [LMR09] the rule designer can specify that for instance a window element that has a button included, but necessarily its parent is not directly the window. Having this extra direct optional attribute does not prevent the meta-model from supporting a regular graph transformation approach.
    - *Restrictions:*
        - The *source* and *target* meta model elements in a mapping relationship cannot be both members of the LHS, RHS or NAC, that is, if a mapping is defined between two meta model elements and one is included in the LHS of a rule, then the other one should be included either in the RHS or the NAC of the same transformation rule.

3.7.6.2.3 LHS

- *LHS:* represents the left-hand side of a graph transformation.
  - *Attributes:*

    - id **(String, required)**: the identifier of the LHS.

    - description **(String, optional)**: description of the LHS. It can be used to document the transformation, in the same way as a regular program is documented.
  - *Relationships:*
    - A LHS aggregates a set of model elements, that is, any element from any UsiXML meta-model.
  - *Restrictions:*
    - A LHS element should aggregate at least one meta-model element.

3.7.6.2.4 RHS

- *RHS:* represents the right-hand side of a graph transformation.
  - *Attributes:*

    - id **(String, required)**: the identifier of the RHS.

    - description **(String, optional)**: description of the RHS. It can be used to document the transformation, in the same way as a regular program is documented.
  - *Relationships:*
    - A RHS aggregates a set of model elements, that is, any element from any UsiXML meta-model.

3.7.6.2.5 NAC

- *NAC:* represents the NAC of a graph transformation.
  - *Attributes:*

    - id **(String, required)**: the identifier of the NAC.

    - description **(String, optional)**: description of the NAC. It can be used to document the transformation, in the same way as a regular program is documented.
  - *Relationships:*
    - A NAC aggregates a set of model elements, that is, any element from any UsiXML meta-model.

3.7.6.2.6 Attribute condition

- *AttributeCondition:* represents a condition defined for an attribute of any meta model element. Note these conditions can appear in meta model elements included in a RHS, LHS or NAC element. This condition can range from simple variables to complex expression. In some graph transformation environments function call with no side effect are also allowed.
  - *Attributes:*

    - id **(String, required)**: the identifier of the attribute condition.

    - attributeName **(String, optional)**: name of the attribute the condition is define on.
  - *Relationships:*
    - An attribute condition aggregates a single expression.
  - *Restrictions:*
    - An attribute condition can include just one expression.

3.7.6.2.7 Expression

- *Expression:* represents an expression. The expression can be mathematical or not. It is usually made of variables, operators and constants.
  - *Attributes:*

    - id **(String, required)**: the identifier of the expression.

    - expression **(String, optional)**: text string representation of the expression.

3.7.7. A Lab Study of the Transformation Meta-model

This section shows a lab study as a proof of concept of the transformation meta-model. We show how the meta-model classes are instantiated to objects that store the needed elements to perform the transformation between two models. The example (based on similar examples used to describe the classes in previous sections) consists in transforming an Abstract Model that represents an interface with several input elements into a Concrete Model with specific widgets.

Figure 17 shows the elements that compose the example of transformation. We start from an Abstract Model that represents a form to create a new customer in a company. We have a container that includes the elements needed to create the customer: two input elements for the name and the surname respectively, and one selection for the customer's marital status (with the items: singles, married, widow). Each input element is transformed into a TextBox but the selection can be transformed into a Radiobox or into a Listbox, depending on usability criteria. Next, we are going to specify this transformation using the transformation meta-model.

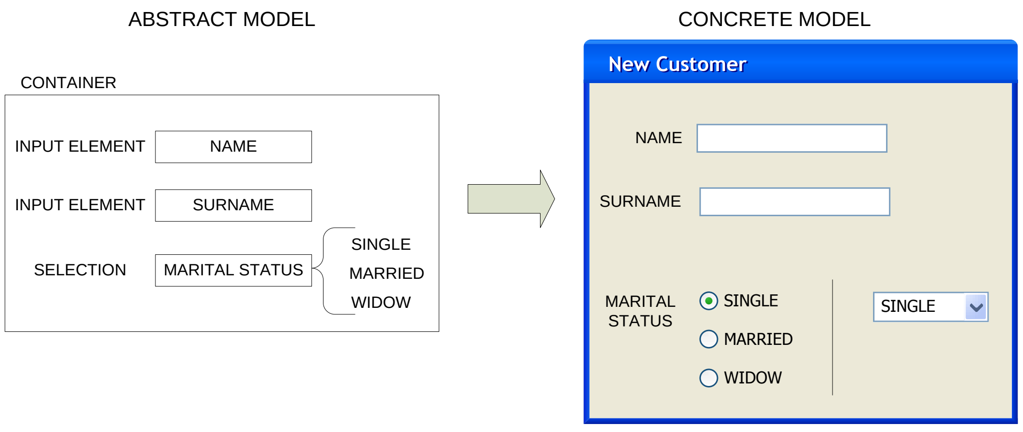


Figure 17. Example of transformation from Abstract Model to Concrete Model

3.7.7.1. Package QOC

This package stores the quality criteria used in the transformation (Figure 10). Figure 18 shows the objects used in the lab study and the relationships among them graphically. Each box represents an object, and the type of each object is labeled in capital letters. Next we are going to explain the meaning of each object in detail.

In the lab study, the system quality depends on the widget to which we transform a selection. The quality is not the same if we transform the selection into a RadioBox or into a ListBox based on the target platform. As follows we discuss the alternatives to improve the quality regarding the selection item.

We represent in each instance of the class DesignSpace a quality target that must be considered in a transformation between two models. In the lab study, this class is instantiated to a target with the label "From

Abstract to Concrete". In our lab study, this DesignSpace is related to one instance of class Question that stores the question to determine the visual aspect of the enumerated elements. The attribute label of Question is "What concrete graphical interactor should be used as target of abstract selection interactors?" and the attribute arch has the value "presentation", since this question is related to presentation issues.

The question has as result several options to satisfy the quality target. We have two instances of the Option class in our lab study. One option has the label "Display in a RadioBox" and the other has the label "Display in a ListBox". The analyst must decide the most suitable option according to a list of criteria that are related to the instance of Question. In our lab study we have used two usability attributes as criteria: Brevity and Information Density. Brevity aims to reduce the cognitive effort of the user, for example, the amount of mouse movements and pressed keys. With regard to Information Density, this attribute aims to reduce the amount of information that the system displays in an interface. Both, Brevity and Information Density are instances of class Criterion. Moreover, these two criteria are related to the same sub-characteristic (Understandability), represented in the transformation model with the class Factor. Options are evaluated against criteria by means of class Assessment. This class is instantiated to 4 objects:

1. The object that relates Radiobox with Brevity has a "strongly support" *value.*
2. The object that relates Radiobox with Information Density has a "strongly denies" *value.*
3. The object that relates Listbox with Brevity has a "strongly denies" *value.*
4. The object that relates Listbox with Information Density has a "strongly support" *value.*

The reason why each relationship between criteria and options has a specific value is explained in the objects of class *Argument*. The reason for each value in our lab study is respectively:

1. A RadioBox improves Brevity because the end-user can select the most suitable option with a single click.
2. A RadioBox decreases Information Density because the system displays all the possible values in the screen, even when the input element is not obligatory.
3. A ListBox decreases Brevity because the end-user has to click on the list to display all the items, use the scroll to select one item, and other click to select a specific item.
4. A ListBox improves Information Density since the items of the list are only displayed when the user click on the widget.

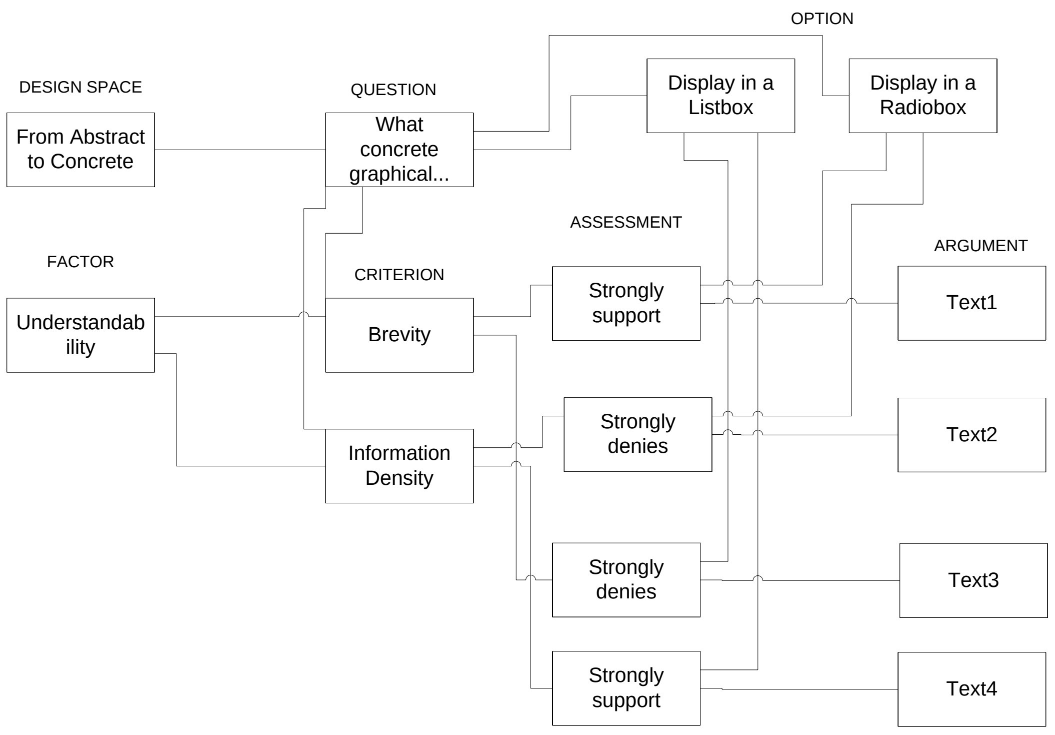


Figure 18. Objects of QOC package in our lab study

3.7.7.2. Package Transformation

This package contains the classes that represent the rules to perform the transformation between two models, as Figure 11 depicts. Figure 19 shows the objects used in the lab study and the relationships among them graphically. Each box represents an object, and the type of each object is labelled in capital letters. Next we are going to explain the meaning of each object in detail.

Transformations are done maximizing and minimizing quality criteria through class *Runtime Configuration*. In our lab study we have two instances of *Runtime Configuration*. One instance has the *name* "Desktop" since aims to optimize the interface for a desktop application. For desktop applications, end-users want to maximize Brevity and minimize Information Density, because, usually, desktop applications do not suffer from space problems in their interfaces. The other instance of *Runtime Configuration* has the name "PDA" since it has the target of optimizing the interface for a PDA. In this case, end-users want to maximize the Information Density and minimize Brevity, since the screen in a PDA is very narrow.

With regard to class *TransformationModel,* it stores the source and target models. In our lab study, the attribute *sourceModelType* has the value Abstract Model and the *targetModelType* has the value Concrete Model. The source and target context is the same. The object of *TransformationModel* is related to an object of *TransformationUnit,* which groups the transformation rules needed to perform the transformation between the source and target models. In our lab study we have three objects of *TransformationUnit*:

1. A TransformationUnit that gathers all the others TransformationUnits and establishes the order in which they must be triggered. The attribute *id* has the value 1, *name* has the value ContainerGraphical, and *orderAmongUnits* has the value sequential since subTransformationUnits are triggered sequentially. This object is not related to questions or options, since it is not related to transformation rules triggered with a choice order and its subTransformationUnits are triggered in a sequential order.
2. A subTransformationUnit of the object with *id* 1 that gathers transformation rules triggered in a sequential order. These transformations are triggered in sequential order independently of questions or

options, therefore, this object is not related to instances of *Question* or *Option*. The attribute *id* has the value 2, *name* has the value ContainerGraphical_SequentialTransformations, and *orderAmongRules* has the value sequential.

3. A subTransformationUnit of the object with *id* 1 that gathers transformation rules one of which will be triggered (choice order). This object is related to the instance of the class *Question* defined in the package *QOC* ("How to display input elements with a limited number of valid entries?"). The attribute *id* has the value 3, *name* has the value ContainerGraphical_ChoiceTransformations, and *orderAmongRules* has the value choice.

The rules that specify how the transformation is performed are stored in instances of the class TransformationRule. All these transformation rules has the same source and target context, therefore, they are related to the same instance of the class ContextModel (these relations have not been depicted in Figure 19 to avoid overloading the diagram). Moreover, these rules use instances of the class MetamodelElements to build source and target elements of the transformation (these relations have not been depicted in Figure 19). Firstly, we are going to describe transformation rules related to the TransformationUnit with id 2 (transformations triggered in a sequential order).

1. An instance of the class *TransformationRule* to transform instances of *AuiClass* into instances of *CuiClass.* The attribute *id* has the value 1 and *name* has the value AuiModel2CuiModel.
2. An instance of the class *TransformationRule* to transform instances of *AuiInteractionUnit* into instances of *CuiInteractionUnit.* The attribute *id* has the value 2 and *name* has the value AuiInteractionUnit2CuiInteractionUnit.
3. An instance of the class *TransformationRule* to transform instances of *AuiObject* into instances of *CuiOject.* The attribute *id* has the value 3 and *name* has the value AuiObject2CuiObject.
4. An instance of the class *TransformationRule* to transform instances of *AuiContainer* into instances of *CuiContainer.* The attribute *id* has the value 4 and *name* has the value AuiContainer2CuiContainerGraphical.
5. An instance of the class *TransformationRule* to transform instances of *AuiInteractor* into instances of *CuiInteractor.* The attribute *id* has the value 5 and *name* has the value AuiInteractor2CuiInteractor.
6. An instance of the class *TransformationRule* to transform instances of *AuiRelationship* into instances of *CuiRelationship.* The attribute *id* has the value 6 and *name* has the value AuiRelationship2CuiRelationship.
7. An instance of the class *TransformationRule* to transform instances of *DataInteractor* into instances of *GraphicalInteractor.* The attribute *id* has the value 7 and *name* has the value DataInteractor2GrInteractor.
8. An instance of the class *TransformationRule* to transform instances of *Input* into instances of *TextField.* The attribute *id* has the value 8 and *name* has the value Input2TextField.

Secondly, we define the transformation rules related to the TransformationUnit with id 3 (transformation rules triggered with a choice order). In this group we have only two rules, one to transform instances of *Selection* into instances of Radiobox and other to transform instances of *Selection* into instances of Listbox. The decision about which transformation rule must be selected depends on the criterion that must be maximized and minimized with regard to the instances of the class *RuntimeConfiguration.* If the system will be used in a Desktop application, then we want to maximize Brevity, and according to the instances of the class *Assessment,* the best option is to generate a Radiobox. On the contrary, if we are developing a system for a PDA, we want to maximize Information Density, and according to the instances of the class *Assessment,* the best option is to generate a Listbox. The description of the two instances of *TransformationRule* is the following:

9. An instance of the class *TransformationRule* to transform instances of *Selection* into instances of *Radiobox.* The attribute *id* has the value 9 and *name* has the value Selection2Radiobox.

10. An instance of the class *TransformationRule* to transform instances of *Selection* into instances of *Listbox.* The attribute *id* has the value 10 and *name* has the value Selection2Listbox.

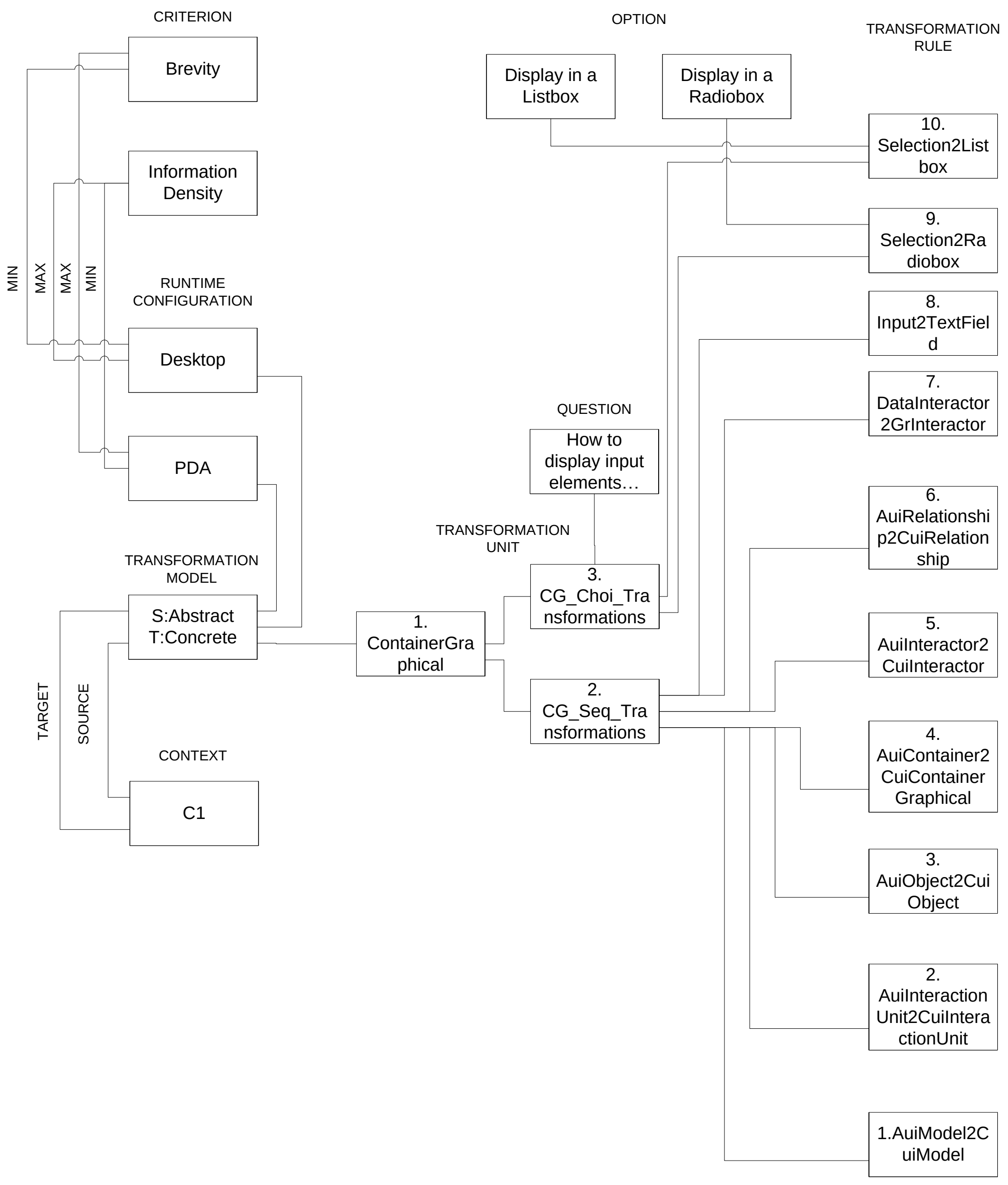


Figure 19. Objects of Transformation package in our lab study

Each one of the instances of *TransformationRule* can be represented in ATL or graphs without distinction. In our lab study we have selected the ATL notation, as we show next:

```
module ContainerGraphical;
create OUT : Concrete from IN : Abstract;

rule AuiModel2CuiModel{
        from
        a:Abstract!AuiModel
        to
        c:Concrete!CuiModel (auiInteractionUnit1 <- a.auiInteractionUnit1)
}
rule AuiInteractionUnit2CuiInteractionUnit{
        from
        a:Abstract!AuiInteractionUnit
        to
        c:Concrete!CuiInteractionUnit (title <- a.title,
                auiInteractionUnit2 <-  a.auiInteractionUnit2,
                auiObject1 <- a.auiObject1)
}
rule AuiObject2CuiObject{
        from
        a: Abstract!AuiObject
        to
        c:Concrete!CuiObject (id <- a.id, label <- a.label, longLabel <- a.longLabel
                , help <- a.help, shortLabel <- a.shortLabel, contextCondition <- a.contextCondition)

}
rule AuiContainer2CuiContainerGraphical{
        from
        a:Abstract!AuiContainer

        to
        c:Concrete!CuiContainer (isSplittable <- a.isSplittable,
                auiContainer2 <- a.auiContainer2),
        g:Concrete!GraphicalContainer ()
}
rule AuiInteractor2CuiInteractor{
        from
        ait:Abstract!AuiInteractor
        to
        cit:Concrete!CuiInteractor ()
}
rule AuiRelationship2CuiRelationship{
        from
        ar: Abstract!AuiRelationship
        to
        cr: Concrete!CuiRelationship (auiInteractor1 <- ar.auiInteractor1,
        auiContainer1 <- ar.auiContainer1)
}
rule DataInteractor2GrInteractor {
        from
        a: Abstract!DataInteractor
        to
        c:Concrete!GraphicalInteractor(),
        g:Concrete!SimpleGrInteractor()

}
rule Selection2Radiobox{
        from
        a:Abstract!Selection
        to
        c:Concrete!RadioBox()

}
rule Selection2Listbox{
        from
        a:Abstract!Selection
        to
        c:Concrete!ListBox()

}
rule Input2TextField {
        from
        a:Abstract!Input
        to
        c:Concrete!TextField(type <- 'text')
}
```

3.7.8. Analysis of the transformation meta-model against the taxonomy of model transformations proposed by Mens et al.

Table 1 presents a summary of the taxonomy of model transformations proposed by Mens et al. [MCG04]. They propose a set of questions (left column of Table 1) and a number of objective criteria (right column of Table 1) to be taken into consideration to provide a concrete answer to the question.

Next, each criterion is briefly explained according to the definitions given in [MCG04] and the proposed UsiXML Transformation meta-model is analyzed against the criterion.

Table 1. Summary of the taxonomy of model transformations proposed by Mens et al. [MCG04]

Question	**Criteria**
What needs to be transformed into what?	Program and model transformation
	Endogenous versus exogenous transformations
	Horizontal versus vertical transformations
	Technological space
What are the important characteristics of a model transformation?	Level of automation
	Complexity of the transformation
	Preservation
What are the success criteria for a transformation language or tool?	Ability to create/read/update/delete transformations (CRUD)
	Ability to suggest when to apply transformations
	Ability to customize or reuse transformations
	Ability to guarantee correctness of the transformations
	Ability to deal with incomplete or inconsistent models
	Ability to group, compose or decompose transformations
	Ability to test, validate, and verify transformations
	Ability to specify generic and higher-order transformations
	Ability to specify bidirectional transformations
	Support for traceability and change propagation
What are the quality requirements for a	Usability and usefulness
	Verbosity versus conciseness

transformation language or tool?	Scalability
	Mathematical properties
	Acceptability by user community
	Standardization
Which mechanisms can be used for model transformation?	Functional programming
	Logic programming
	Graph transformation

- **Program and model transformation**: our transformation meta-model supports model transformations. According to [MCG04], model transformations encompass program transformations. Hence, our transformation meta-model also supports program transformations, for instance a model-to-code transformation. However, we have not tested yet this kind of transformations.
- **Endogenous versus exogenous transformations**: endogenous transformations are transformations between models expressed in the same language. Exogenous transformations are transformations between models expressed using different languages. Our transformation meta-model supports both, endogenous and exogenous transformations. Reflection and translation are endogenous. Abstraction and reification are exogenous.
- **Horizontal versus vertical transformations**: a horizontal transformation is a transformation where the source and target models reside at the same abstraction level. In a vertical transformation, the source and target models reside at different abstraction levels. Our transformation meta-model supports both, horizontal and vertical transformations. Reflection and translation are horizontal. Abstraction and reification are vertical.
- **Technological space**: a distinction is made on whether the source and target models belong to one and the same or to different technological spaces. In our transformation meta-model, the concept of technological space can be related to the concept of context of use. Hence, we support transformations between models of the same or different technological spaces. Reflection, abstraction, and reification are transformations in which the technological space is not changed. Translation is a transformation that adapts a model from one technological space to another.
- **Level of automation**: there are transformations that can be automated and transformations that need to be performed manually. Our transformation meta-model allows to define core transformation rules which are not executable per se, but each core transformation rule must have at least one rule representation which is executable, for instance using ATL and/or graphs.
- **Complexity of the transformation**: our transformation meta-model allows to define from small transformations such as reflections, to heavy-duty transformations such as model-to-code generators.
- **Preservation**: each transformation preserves certain aspects of the source model in the transformed target models. We need to perform a deeper analysis of this criterion in order to identify the aspects that should be preserved in the different types of transformations. Preservation is not currently supported.
- **Ability to create/read/update/delete transformations (CRUD)**: this ability is supported in the transformation meta-model.
- **Ability to suggest when to apply transformations**: our transformation meta-model supports this ability relating transformation rules or transformation units to option designs that can be analyzed against different criteria in order to select the one that will be executed.

- **Ability to customize or reuse transformations**: our transformation meta-model supports the reuse of transformation rules and transformation units.
- **Ability to guarantee correctness of the transformations**: transformations can be syntactically correct: given a well-formed source model, a transformation guarantee the production of a well-formed target model. Transformations can also be semantically correct: the target model has the expected semantic properties. Currently, our transformation meta-model does not support this ability.
- **Ability to deal with incomplete or inconsistent models**: mechanisms for inconsistency management should be provided in order to deal with ambiguous, incomplete or inconsistent models. Currently, our transformation meta-model does not support this ability.
- **Ability to group, compose, and decompose transformations**: in our meta-model, transformation models are aggregations of transformation units which, in turn, can aggregate other transformation units and transformation rules. We also provide mechanisms to specify the order in which sub-transformation units and transformation rules must be applied.
- **Ability to test, validate, and verify transformations**: currently, our transformation meta-model does not support this ability.
- **Ability to specify generic and higher-order transformations**: if it is possible to represent transformations as models, it is possible to apply transformations to these models, thus achieving a notion of higher-order transformations. Since we are providing a meta-model for transformations, transformations will be specified as models, and hence, it will be possible to apply transformations to transformation models.
- **Ability to specify bidirectional transformations**: bidirectional transformations can be used in two directions: to transform the source model(s) into target model(s), and the inverse transformation to transform the target model(s) into source model(s). Our transformation meta-model does not give support to bidirectional transformations.
- **Support for traceability and change propagation**: to support traceability, the transformation language or tool needs to provide mechanisms to maintain an explicit link between the source and target models of a model transformation. To support change propagation, the transformation language or tool may have an incremental update mechanism and a consistency checking mechanism. Currently, our transformation meta-model does not support this ability. However, a mapping meta-model will be provided to support traceability.
- **Usability and usefulness**: the language or tool should be useful, which means it has to serve a practical purpose. On the other hand, it has to be usable too, which means that it should be intuitive and efficient to use. These properties have not been tested yet.
- **Verbosity versus conciseness**: conciseness means that the transformation language should have as few syntactic constructs as possible. From a practical point of view, however, it often requires more work to specify complex transformations. Hence, the language should be more verbose. We can considerer that our transformation meta-model will lead to verbose transformation models since transformations rules can be expressed using different rule representations.
- **Scalability**: the language or tool should be able to cope with large and complex transformations or transformations of large and complex software models. The aim of our transformation meta-model is to deal with transformation of complex interactive systems.
- **Mathematical properties**: if the transformation language or tool has a mathematical underpinning, it may be possible, under certain circumstances, to prove theoretical properties of the transformation such as termination, soundness, completeness (syntactic and semantic), correctness, etc. Our transformation meta-model does not support this ability.

- **Acceptability by user community**: the transformation meta-model have not been tested by the user community yet.
- **Standardization**: the transformation meta-model, as well as other meta-models of the UsiXML project will be subject of standardization efforts.
- Regarding the last 3 criteria, **functional programming**, **logic programming**, and **graph transformation**, we can say that the transformation meta-model allows a transformation rule to have different rule representations. Right now we have considered ATL and graphs for the rule representations. However, other representations (functional or logical) could also be added.

3.8. Workflow

The UsiXML workflow model describes the flow of tasks and information that are passed from one worker to another, according to a set of procedural rules. This model is decomposed into processes that are in turn refined into tasks. Tasks are related to produced/consumed resources and to job definitions of the organization. As such, the UsiXML workflow model has four aspects: functional, behavior, organizational and informational aspect. These aspects are already supported by a set of UsiXML models:

- Functional aspect describes the interactive tasks as viewed by the end user interacting with the system. This aspect is supported by the task model (Section 5.2);
- Behavior aspect describes logical and temporal relationships between tasks. This aspect is also supported by the task model;
- Organizational aspect describes a decomposition of the organization structure into organizational units (e.g., a service) that are responsible for defining the jobs and their responsibilities. This aspect is supported by the organization model;
- Informational aspect describes the classes of objects manipulated by a user while interacting with a system. This aspect is supported by the domain model (Section 5.4);

The UsiXML workflow model links the task model, the domain model, and the organization model by matching the elements of these models to each other. The workflow model defines the relationships between tasks (e.g. sequence of tasks), the relationships between task and domain (e.g. one task produces one data) and the relationships between task and organization (e.g. one task is performed by one organization). Note that, the UsiXML workflow model will not replace the task model because the task model describes, opposed to workflow models, the hierarchical logical structures of one task (the root) and the relationships between its sub-tasks. However, the workflow model describes the relationships between the root tasks of the User Interface. Another note is that, in the current version of the UsiXML, the mapping model describes a set of pre-defined relationships that allows the matching of elements from the UsiXML models (Section 5.10). For example, the relationship Manipulate matches a task to a domain concept. The UsiXML mapping model does not link only the elements of the task model, the domain model, and the organization model. It defines also the relationships between the other UsiXML models. For example: the relationship Is Reified By matches the abstract user interface model to concrete user interface model. Thus, the workflow model is a part of the mapping model. But, the UsiXML workflow model provides a more interesting graph view of the flow of tasks, the flow of data and the involved organization. For this reason, the UsiXML workflow model needs to be based on a sound model, which helps to provide an understandable representation of the different workflow aspects.

The UsiXML workflow meta-model is based on the BPMN (Business Process Modeling Notation) meta-model [OMG10]. Indeed, the BPMN is an OMG standard model for workflow description. This standard defines a common graphical notation to describe workflow aspects. It separates the business information from technical information and provides a correspondence to an execution model. This standard is close to UML class diagram. The BPMN is associed with a specific graphical notation.

3.8.1. Overview

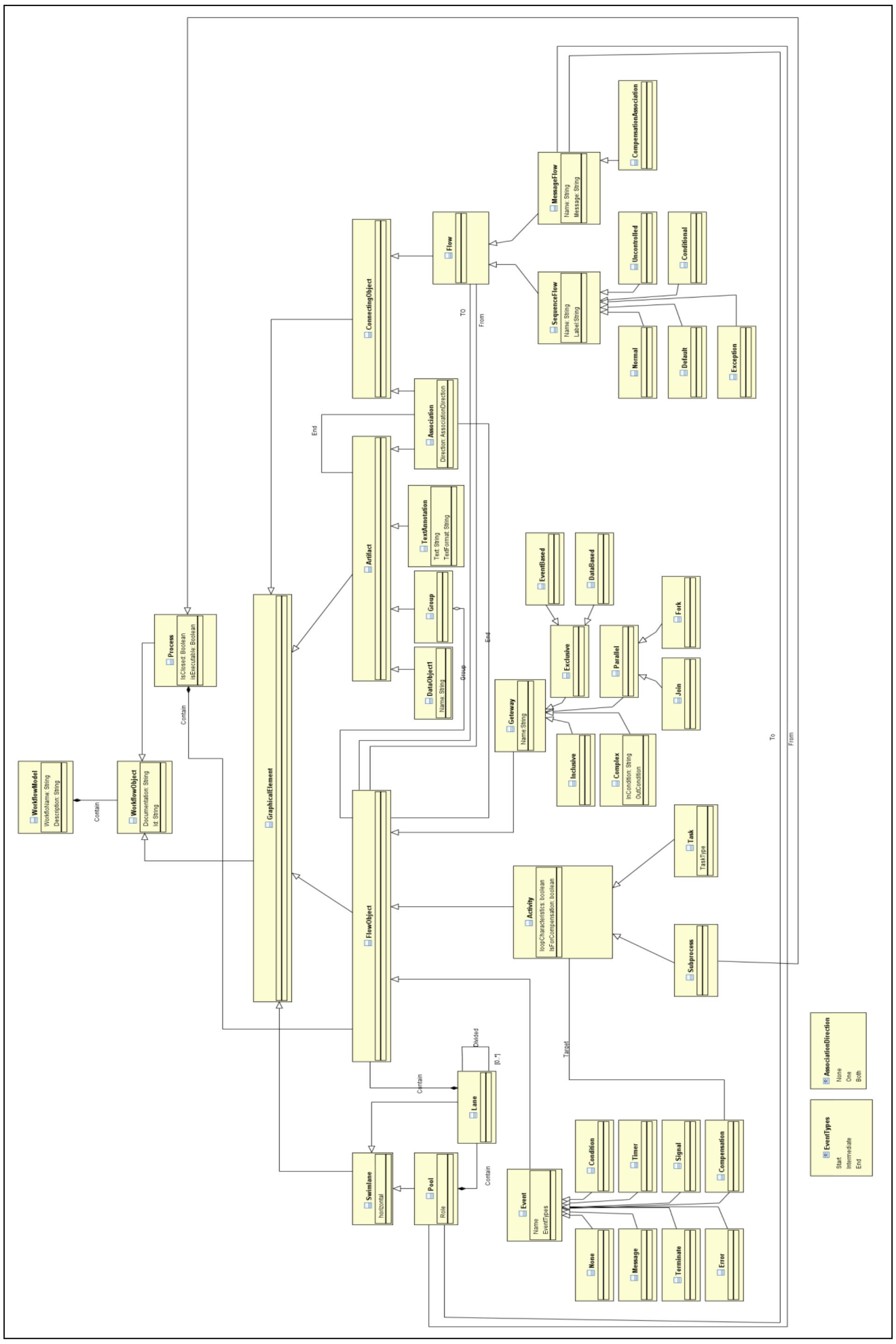


Figure 20 - UsiXML workflow Meta-model based on the BPMN [OMG10]

Figure 20 shows the UsiXML workflow meta-model that is based on the BPMN meta-model. The main classes of this meta-model are defined in the following section. Note that, the definition of these classes is from the OMG specification [OMG10].

- *Activity:* An Activity represents the work that is performed in a workflow. It can be a Task or a sub-process.
  - *Attributes:*
    - loopCharacteristics (**boolean**): identifies whether this Activity is intended for the purposes of compensation
    - IsForCompensation (**boolean**): indicates if the Activity may be performed once or may be repeated.

- *Artifact:* represents the graphical element that provides additional information about the Process or elements within the Process. An artefact can be a DataObject, a Group, a TexteAnnotation, or an Association.
  - *Attributes:*

  No specific attribute

- *Association:* An Association is used to link an artefact with a flow object.
  - *Attributes:*

    Direction (**AssociationDirection):** Indicates whether an association is navigable or not.

- *AssociationDirection:* Association Direction kind is an enumeration type of the direction of the Associations. Association Direction kind is an enumeration of the following values:
  - *None*
  - *One*
  - *Both*

- *CompensationAssociation:* Compensation Association occurs outside the normal flow of the Process and is based upon a Compensation Event that is triggered through a failure.
  - *Attributes:*

    No specific attribute

- *CompensationEvent:* indicates that compensation will be lunched. If an Activity is identified, and it was successfully completed, then that Activity will be compensated.
  - *Attributes:*

  No specific attribute

- *ConditionEvent:* *represents an* event that is triggered when a condition is satisfied.
  - *Attributes:*

    No specific attribute

- *ConditionFlow:* represents a Sequence Flow with a condition that is evaluated at runtime to determine whether or not the Sequence Flow will be used
  - *Attributes:*

No specific attribute

- *ConnectingObject:* is a relationship that relates two elements or sets of elements workflow elements. A connectingObject can be Connecting flow or an association.
  - *Attributes:*

  No specific attribute

- *DataBasedGateway:* represents a branching point where Alternatives are based on the process data.
  - *Attributes:*

    No specific attribute

- *DataObject:* a Data Object provides information about what Activities manipulated and/or what they produce.
  - *Attributes:*
    - name (**String**) : The name of the data object

- *DefaultFlow:* expresses the default branch to be chosen if all the conditions evaluate to false. This flow is used with the Data-Based Exclusive Gateways or Inclusive Gateways.
  - *Attributes:*

    No specific attribute

- *ErrorEvent:* indicates that a generated Error.
  - *Attributes:*

  No specific attribute

- *Event:* indicates that signals that a situation has occurred and for which a response is necessary.
  - *Attributes:*
    - name (**String**): The name of the event
    - EventType **(EventTypes)**: The type of the event

- *EventBasedGateway:* represents a branching point where Alternatives are based on an Event that occurs at that point in the Process (e.g. the receipt of a Message).
  - *Attributes:*

    No specific attribute

- *EventTypes:* is an enumeration type that specifies the kind of events. There are three types of Events, based on when they affect the flow:
    - Start Event indicates where a Process will start
    - Intermediate Events occur between a Start Event and an End Event. They will affect the flow of the Process, but will not start or (directly) terminate the Process
    - End Event indicates where a Process will end

- *ExceptionFlow:* expresses that the flow deviates from the normal flow. For example, A error event, can initiate exception flow. After triggering at least one activity, his exception flow may lead to a stop event or may rejoin the normal flow.
  - *Attributes:*

    No specific attribute

- *ExclusiveGateway:* represents alternative paths within a process where only one path can be executed (XOR-split).
  - *Attributes:*

    No specific attribute

- *FlowConnecting:* represents a directional link between the flow objects. This class represents an abstraction of two elements (Sequences, Messages).
  - *Attributes:*

  No specific attribute

- *FlowObject:* represents a directional link between elements in a Process. This class is an abstraction of three core elements (Events, Activities, and Gateways).
  - *Attributes:*

  No specific attribute

- *Fork:* fork refers the dividing of a path into two or more parallel paths (AND-Split). It expresses that activities can be performed concurrently, rather than sequentially.
  - *Attributes:*

    No specific attribute

- *Gateway:* A Gateway is used to control the divergence and convergence of Sequence Flows in a workflow.
  - *Attributes:*
    - name (**String**) : The name of the gateway.

- GraphicalElement: represents the graphical element that is used in the workflow model to describe a process.
  - *Attributes:*

  No specific attribute

- *Group:* A Group is a grouping of flow objects that are within the same Category. This type of grouping does not affect the Sequence Flows within the Group.
  - *Attributes:*

    No specific attribute

- *InclusiveGateway:* represents alternative paths within a Process where one or more paths may execute. Unlike the Exclusive Gateway, all condition expressions are evaluated. The true evaluation of one condition expression does not exclude the evaluation of other condition expressions. All Sequence Flow with a true evaluation will be traversed by a Token.

- *Attributes:*

  No specific attribute

- *Join:* join refers to the combining of two or more parallel paths into one path (AND-Join or synchronization).
  - *Attributes:*

    No specific attribute

- *Lane:* A Lane is a sub-partition within a Pool. Lanes are used to organize and categorize Activities.
  - *Attributes:*

    No specific attribute

- *MessageEvent:* A Message arrives from a Participant (pool).
  - *Attributes:*

    No specific attribute

- *MessageFlow:* A Message Flow is used to show the flow of Messages between two Participants (Pools).
  - *Attributes:*
    - name (**String**): The name of the Message Flow.
    - Message (**String**): The message of the Message Flow.

- *NoneEvent:* expresses an event that does not have a defined trigger.
  - *Attributes:*

  No specific attribute

- *NormalFlow:* Normal flow refers to paths of Sequence Flow that originates from a Start Event and continues through activities via alternative and parallel paths until it ends at an End Event
  - *Attributes:*

    No specific attribute

- *ParallelGateway:* A Parallel Gateway is used to show the joining of multiple Sequence Flows.
  - *Attributes:*

    No specific attribute

- *Pool:* A Pool is the graphical representation of a Participant (user of the User Interface).
  - *Attributes:*
    - role (**String**) : The role of the participant.

- *Process:* Contains Flow objects (Activities, Events, Gateways, and Sequence Flow) that adhere to a finite execution of the process.
  - *Attributes:*

- isClosed (**Boolean**): specifies whether interactions, such as sending and receiving Messages and Events, not modeled in the Process can occur when the Process is executed or performed.
- IsExecutable (**Boolean**): specifies whether the Process is executable.

- *Sequence Flow:* A Sequence Flow is used to show the order that Activities will be performed in a workflow
  - *Attributes:*
    - name (**String**) : The name of the Sequence Flow.
    - label (**String**) : The label of the Sequence Flow.

- *SignalEvent:* represents a signal arrives that has been broadcast from another Process. Note that the Signal is not a Message, which has a specific target for the Message.
  - *Attributes:*

    No specific attribute

- *Sub-Process:* A Sub-Process is a compound activity that is included within a Process.
  - *Attributes:*

  No specific attribute

- *Swimlane:* Swimlane describes the organizational aspect of a workflow. This represents an abstraction of two core elements (Pool, and Lines).
  - *Attributes:*

  No specific attribute

- *Task:* A Task is an Activity that is included within a Process. It models the UsiXML root tasks of the User Interface.
  - *Attributes:*
    - TaskType **(TaskTypes)**: The type of the task (Section 5.2)

- *TerminateEvent:* indicates that all Activities in the Process should be immediately ended. This includes all instances of multi-instances. The Process is ended without compensation.
  - *Attributes:*

  No specific attribute

- *TextAnnotation:* Text Annotations are a mechanism for a modeler to provide additional text information for the reader of a workflow model.
  - *Attributes:*

    No specific attribute

- *TimeEvent:* A specific time-date or a specific cycle (e.g., every Friday at 9am) can be set.
  - *Attributes:*

  No specific attribute

- *UncontrolledFlow:* Uncontrolled flow refers to flow that is not affected by any conditions or does not pass through a Gateway.

- *Attributes:*

  No specific attribute

- *WorkflowModel:* Workflow model describes the tasks and information that are passed from one worker to another, according. A workflow model is made up of WorkflowObjects.
  - *Attributes*
    - WorkflowName **(String):** the name of the workflow model
    - description **(String):** the textual description of the workflow model.

- *WorkflowObject:* A workflow object is a constituent of a workflow model. This class represents an abstract generalization class that is specialized in the meta-model.
  - *Attributes:*

  No specific attribute

3.8.2. Example

Figure 21 gives an example of a UsiXML workflow model expressed using BPMN. In this workflow model, the rounded-corner rectangles represent the root tasks of the User Interface (e.g. Create a waypoint, Transmit waypoint). The circles represent the events that are triggered by tasks, or the event that trigger the tasks. The diamond shapes represent the gateways (e.g. data based gateway-Xor) that control the divergence and convergence of the workflow objects. The solid lines with solid arrowheads represent the sequence flows that show the order (the sequence) of the tasks performance. In turn, the dotted lines with line arrowheads represent the associations relationship that are used to associate artifact (data, text, and other) with flow objects; the graphical container represents the pool that is used to express a user of the User Interface. finally, the folded-corner rectangle represents the data object (e.g. waypoint) that model the information aspect of the workflow.

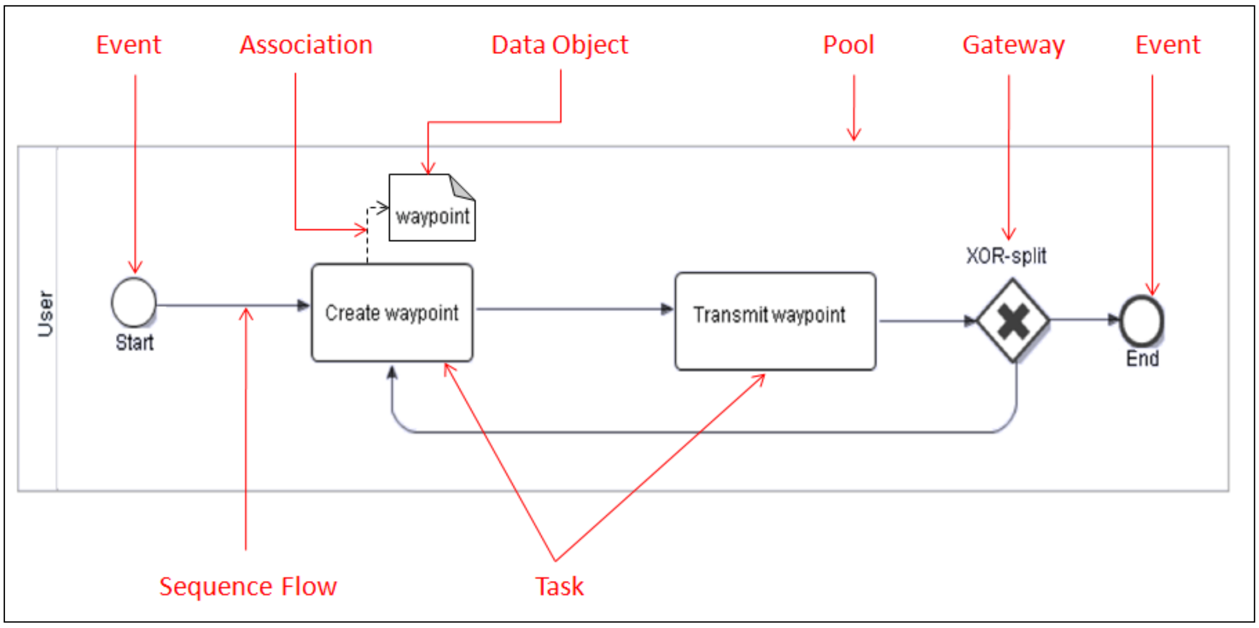


Figure 21. UsiXML workflow Model Example

3.9. Quality

3.9.1. Overview

Figure 22 shows the Quality Meta-Model.

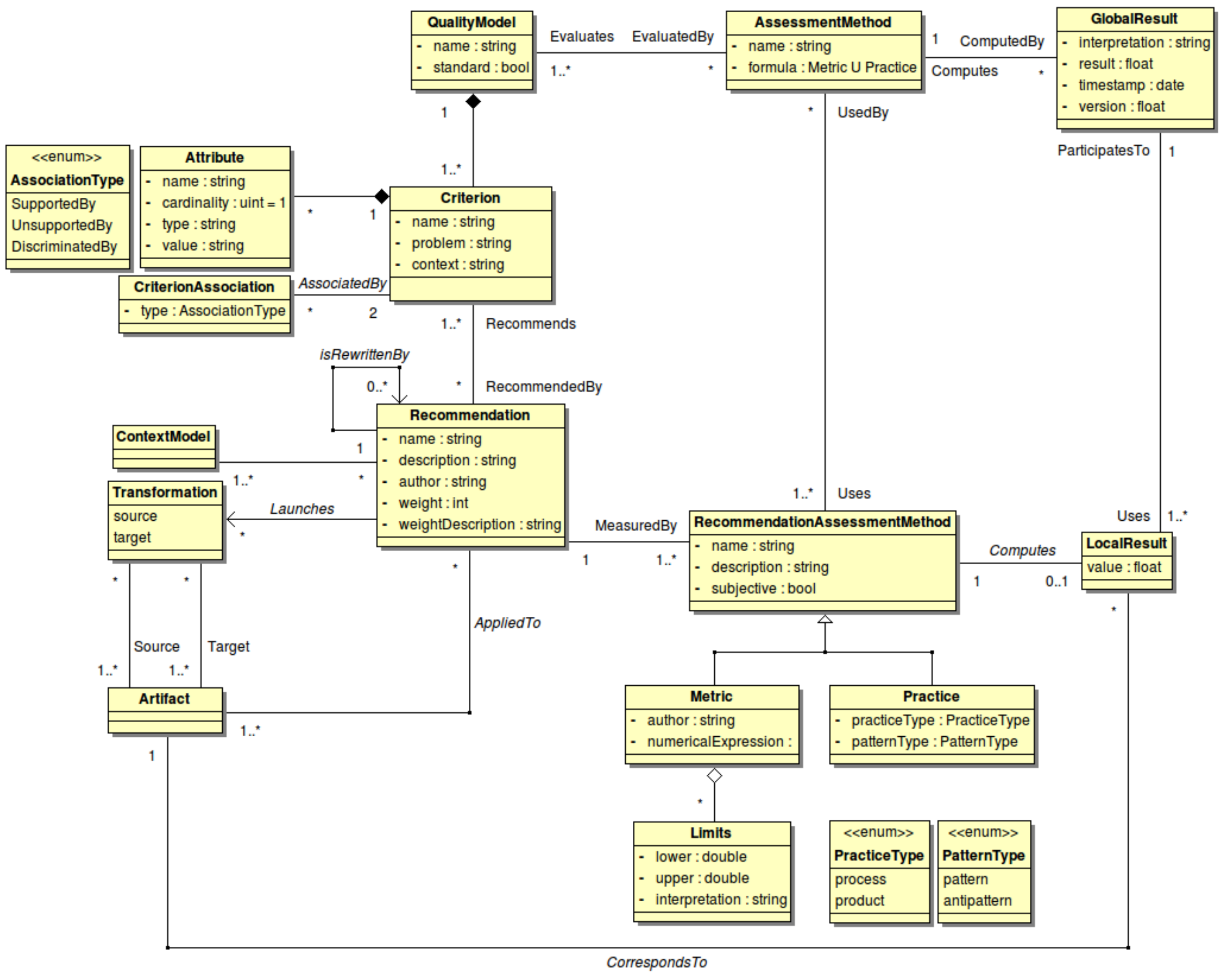


Figure 22 - Quality Meta-Model overview

3.9.2. Summary

Different quality models have been proposed in the literature. McCall's hierarchical quality model [MRW77] focuses on product quality, organizing it in two views: the external view for end-users and the internal view for developers. Boehm's model [BBKLMM78] adds a third level named primitive characteristics to deal with metrics and evaluation. The ISO/IEC 9126 standard series divides metrics into internal, external and quality-in-use. This quality-in-use, also called usability or perceived quality, has been the main focus of the HCI community. Usability has evolved through standards such as the ISO 9241-110 [ISO9241], ISO/IEC 9126-1 [ISO9126] and ISO/IEC 25010 [ISO25010] among others. As a synthesis, Seffah encompasses most of the usability works in

QUIM [SDKP06]. However Software Engineering quality models are more than usability. They deal with other important aspects of general quality in the whole System Development Life Cycle. ISO standards deal also with these aspects. To cover them, different quality Meta-Models have been proposed such as [KSC09] for data quality, [MD96] as a quality Meta-Model for MDE, or [D96] that defines a five step process for building product-specific quality models. However, whilst several quality models exist in Software Engineering, most of them are oriented to evaluating source code or final products and not models or modeling activities. Other models don't deal with evaluation aspects (evaluation methods, results...) or they just miss the different quality perspectives.

The Quality Meta-model has been designed to overcome these problems. Moreover, it respects the following four basic principles:

1. The Quality Meta-model must be generic and domain independent (not limited to HCI).
2. The Quality Meta-model must be independent of the way in which the measurement is done.
3. The Quality Meta-model must be independent of the type of Criteria which compose the Meta-model.
4. The Quality Meta-model must be independent of the way in which argumentation is done (not limited to the QOC Meta-model).

3.9.3. Quality perspectives

The proposed Quaity Meta-Model (Figure 22 and Figure 23) has been designed to cover the needs of both Software Engineering and HCI. Quality can be expressed regarding four different perspectives [C02]:

- *Expected Quality*, or the quality the client needs. It is defined through the specification of the SUS.
- *Wished Quality* is the degree of quality that the quality expert wants to achieve for the final version of the SUS. It is derived from the Expected Quality.
- *Achieved Quality* is the quality obtained for a given implementation of the SUS. Ideally, it must satisfy the Wished Quality.
- *Perceived Quality* is the perception of the results by the client, once the SUS has been delivered.

As stated in [SAC02], these four perspectives can be related to the Systems Development Life Cycle by three dimensions. These dimensions are the Specification (related to the *Expected* and *Wished Qualities*), Implementation (related to the *Achieved Quality*) and Use (related to the *Perceived Quality*). The Quality Meta-Model deals with these four perspectives as shown in Figure 23. Here, the *System* entity represents the product to consider. *SysEval* represents a specific evaluation for that product. The four quality perspectives are four different uses of the same quality model. The attribute *standard* means that, when true, the quality model is not linked to *System* and *SysEval* as it only represents a quality standard such as ISO9241-110 or QUIM. In other words, the quality of these standards is not defined in terms of a product. Some internal parts of the Quality Meta-Model are not necessarily defined when the attribute *standard* is true.

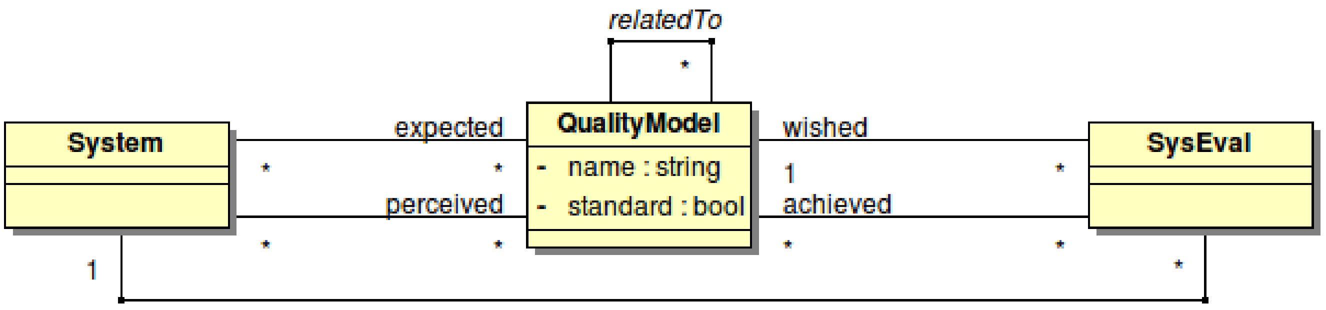


Figure 23 - Quality perspectives in the Quality Meta-Model

3.9.4. The Meta-Model

Figure 22 shows the *Quality Meta-Model* in detail. A quality model is composed of criteria, that can be recursively decomposed into subcriteria through the meta-class *CriterionAssociation*. Different *Recommendations* can be specified for each *Criterion*. A *Recommendation* is a positive assessment that characterizes Criteria. We can specify a weight for each *Recommendation* to define which of them are more important than others for the considered system. This allows designers to adjust the global quality precisely. Evaluations can be performed through *AssessmentMethods* that are specified by *Metrics* and/or *Practices*. In the first case, the measure is given by a *Result* that can be comprised between some *Limits* when defined. In the case of *Practices*, the *Result* represents if a practice has been followed with a value of 100% or not (0%). The value of the *Result* can be any intermediary percentage as well. Note that a *Practice* can be either a pattern or an anti-pattern, applied at the process level, or on a product. *Metrics* and *Practices* are directly evaluated on *Artifacts* through *Recommendations*. An *Artifact* can be no matter what element of the Software Development Life Cycle, such as code, classes of a model or the model itself.

Once a quality standard has been defined through Criteria, the Quality Meta-model can be reused with the association *relatedTo*, and extended with several classes such as *AssessmentMethods*, *Transformations* or *Artifacts*, to represent the four quality perspectives. For instance, *Metrics* can be defined in order to obtain some desired values (*Wished Quality*). The importance of every *Recommendation* can be customized using *Weights*. Then, evaluations of the current quality of the SUS can be performed. When a *Result* of an evaluation does not satisfy the expectations of the quality expert, this is, the *Achieved Quality* does not satisfy the *Wished Quality* (for instance, the value for a *Metric* is not within the desired *Limits*), the designer needs to increase the quality. This can be done by setting a *Transformation* or a set of *Transformations*. These *Transformations* are performed on the related *Artifacts* on which the *Result* has been previously calculated. Iterations can be done until the desired values defined by the quality expert (*Wished Quality*) are reached. *GlobalResult* holds the general quality of a SUS at a given moment. The difference between *GlobalResults* and *LocalResults* is explained in the next section.

3.9.5. Objects, Methods and Results, Global Quality vs Local Quality

Figure 24 shows the different subsets of the Quality Meta-model regarding Global and Local quality levels. To explain these levels, three vertical columns make explicit what **Objects** are being measured at the current level, which is the element responsible of the measurement **Method**, and the quality level of the **Result** for such object.

We define these levels as follows:

- The **Global Quality Level** is the group of Objects, Methods and Results directly focused on the general quality of a SUS.
- The **Local Quality Level** is the group of Objects, Methods and Results focused on the quality of a given *Criterion* (and then, all the associated *Recommendations*).

The global quality of a SUS at a given moment according to a Quality Model is represented by the *GlobalResult* meta-class, and it is directly computed following the formula described in an *AssessmentMehtod*. At the Local Quality Level, the *LocalResult* meta-class represents partial contributions to the quality of the SUS. *Criteria* is evaluated through *Recommendations* by *RecommendationAssesmentMethods,* each of them providing one *Result.* All these results are pondered later at the Global Level. The importance of each *Recommendation* is specified by *Weights* that can be used by the quality expert in the *AssessmentMethod* formula.

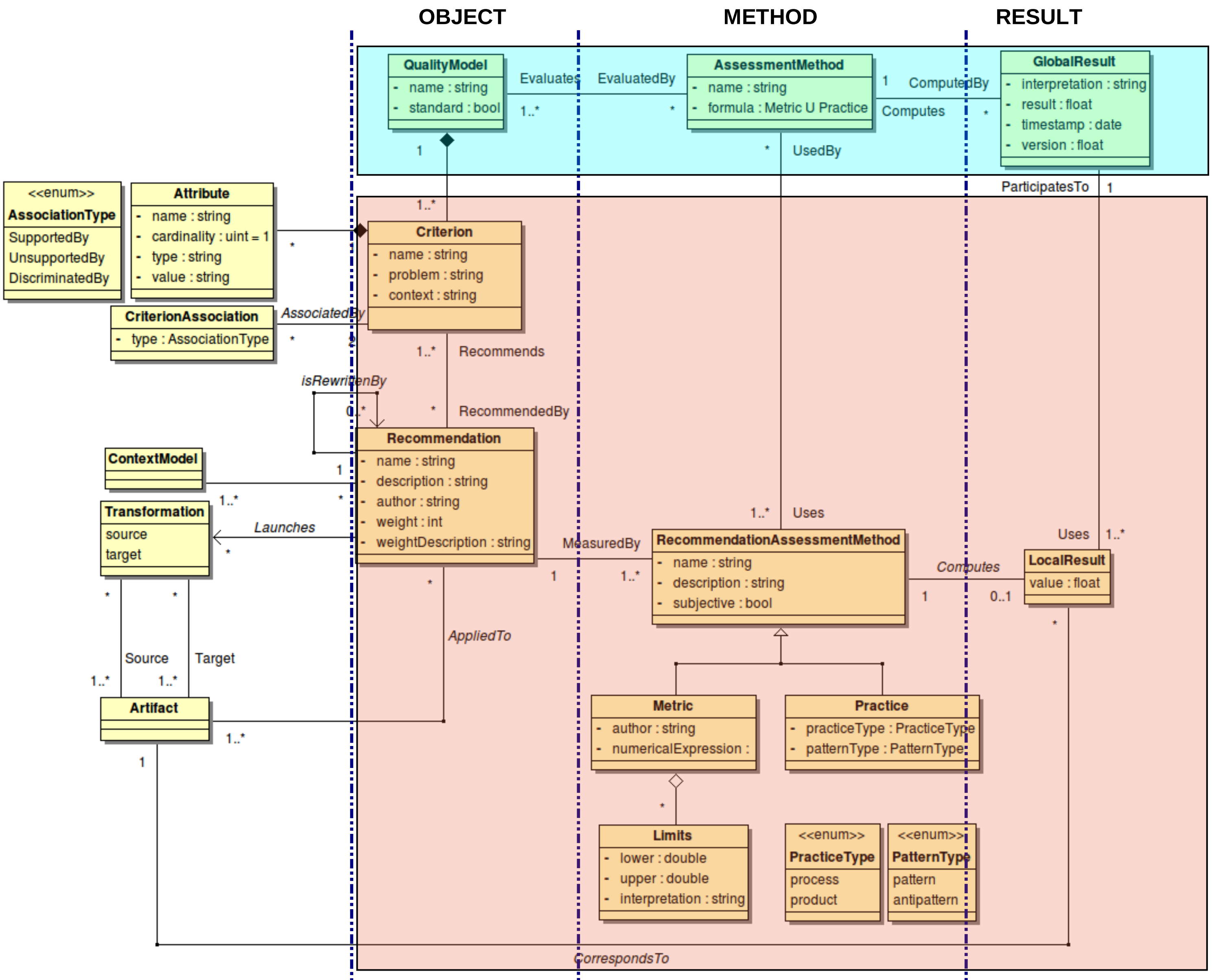


Figure 24. Global and Local Quality levels.

🟦 **Global level**. The global quality of a SUS at a given moment according to a Quality Model is represented by the *GlobalResult* meta-class, and it is directly computed following an *AssessmentMehtod*.

🟥 **Local level**. The *LocalResult* meta-class represents partial contributions to the quality of the SUS. *Criteria* is evaluated through *Recommendations* by *RecommendationAssesmentMethods,* each of them providing one *Result.* All these results are pondered later at the Global Level.

3.9.6. Classes of the Quality Meta-model

Next, we explain the meaning of the classes of the Quality meta-model:

- *QualityModel:* The *QualityModel* meta-class defines the representation of a Quality Model.
  - *Attributes:*
    - name **(String)**: Specifies the name of the Quality Model.
    - standard **(Boolean)**: Specifies whether the current instance of the Quality Meta-Model represents a quality standard or not. If *true,* the quality model represents a standard such as ISO 9241-110. This means that the model is composed only of instances of *QualityModel*, *Criteria*, *Attribute* and *CriterionAssociation* meta-classes.
- *Criterion:* The *Criterion* meta-class describes how the Quality Meta-Model is composed. A quality model is composed of criteria, that can be recursively decomposed into subcriteria as well through the *CriterionAssociation* class. This representation allows to instantiate different standards from different communities such as the Software Engineering community (for instance to evaluate the quality of the source code) or the HCI community (for example, instantiating the four layers of QUIM[17].)
  - *Attributes:*
    - name **(String)**: Defines the name of the *Criterion.*
      - Example: Usability for the task.
    - problem **(String)**: Defines the problem the *Criterion* is dealing with.
    - context **(String)**: Specifies the context in which the Criterion applies.
- *Attribute:* Definition
  - *Attributes:*
    - name **(String)**: It allows to specify one or more attributes for a Criterion.
    - cardinality **(Unsigned Int)**: Defines the cardinality of the attribute. By default, the cardinality is one.
    - type **(String)**: Defines the type of the attribute.
    - value **(String)**: Holds the value of the attribute.
- *CriterionAssociation:* The *CriterionAssociation* is an abstract element that defines the relationship of the *Criterion* accordingly to the definition of the Quality Model.
  - *Attributes:*
    - type **(AssociationType)**: Defines the type of the association. This allows to define how different Criteria are related.
  - Possible values: *SupportedBy*, *UnsupportedBy*, *DiscriminatedBy*. A *Criterion* can support other criteria (for instance, in QUIM a factor at the Factor level is supported by criteria from the Criteria level). It can be discriminated by other *Criterion*, typically when two criteria are in conflict, or the relationship can be unsupported when two Criteria are not in conflict but there is no support between them.
- *Recommendation:* A *Recommendation* is a positive assessment that corresponds to one or more criteria. For instance, the *Recommendation* says that good quality can be achieved by maximizing the number of criteria that are satisfied by a given UI. Figure 5 shows how different *Metrics* are used for the same *Recommendation.* A

Recommendation can be decomposed or rewritten in sub-recommendations through the *isRewrittenBy* association.

- *Attributes:*

  - name **(String)**: Defines the name of the *Recommendation.*

  - description **(String)**: Explains the *Recommendation.*

  - author **(String)**: Defines the name of the author of the *Recommendation*, to keep trace of the different *Recommendations* each quality expert has done.

  - weight **(Integer)**: Defines the current *weight* of a *Recommendation.* The *weight* allows the quality expert to model how important a *Recommendation* is with regard to others.

  - weightDescription **(String)**: Explains how the *weight* is interpreted.

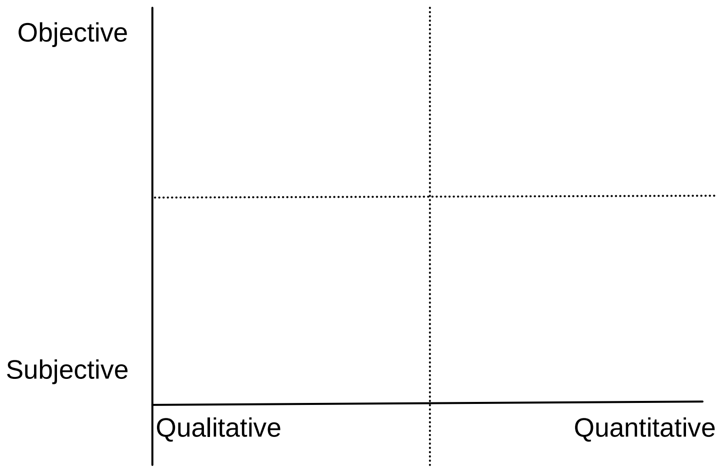
- *RecommendationAssessmentMethod:* This class represents the way in which the quality expert or the system itself can determine if a *Recommendation* is accomplished or not. A *RecommendationAssessmentMethod* is specialized in *Metrics* or *Practices*. It can be subjective or objective.

  - *Attributes:*

    - name **(String)**: Defines the name of the Metric or Practice to be used.

    - description **(String)**: Explains the Metric or Practice, describing the formula and its different elements in the case of a Metric, or what does the Practice involve and how to know if it has been followed or not.

    - subjective **(Boolean)**: Explains whether the measurement is subjective (true) or objective (false). Note that even subjective evaluations can be measured quantitatively (for instance by *Metrics*) or qualitatively (for instance by a *Practice*). The attribute *subjective* makes explicit this distinction and allows quality experts to cover both dimensions as depicted in the next figure:



- *Metric:* Express how to compute a numerical value for a given *Artifact. Metrics* are associated to *NumericalResults*.

  - *Attributes:*

    - author **(String)**: The author of the metric.

    - numericalExpression **(String)**: Defines the associated formula for the metric.

- *Limits:* Holds the desired values for a given metric.
    - *Attributes:*

      - lower **(Double)**: Defines the minimum value the metric is desired to achieve.

      - upper **(Double)**: Defines the maximum value the metric is desired to achieve.

      - interpretation **(String)**: Explains how to interpret ate the limit values.
- *Practice:* The *Practice* meta-class represents *Practices*, i.e., proven processes or techniques that organizations or persons have found to be productive and useful to ensure a good level of quality (Good Practices), or unproductive and unusable (Bad Practices). "Design patterns" are an example of the first one, whilst "Spaguetti code" is an example of the second one.
    - *Attributes:*

      - practiceType **(PracticeType)**: Defines if the *Practice* is applicable to a *Process* or a *Product.*

        - Possible values: process, product.

      - patternType **(PatternType)**: Defines if the *Practice* represents a *Pattern* or an *Antipattern.*

        - Possible values: pattern, antipattern.
- *LocalResult:* Holds the result of an AssessmentMethod.
    - *Attributes:*

      - value **(Float)**: In the case of *Metrics,* the value represents the result of the computation of the *numericalExpression* of the *Metric.* In the case of a *Practice,* the value attribute represents the percentage in which a *Practice* is satisfied.
- *AssessmentMethod:* This meta-class specifies how to compute *Metrics* and *Practices* together. The global quality of a SUS is computed through *AssessmentMethods.*
    - *Attributes:*

      - name **(String)**: Defines the *AssessmentMethod* name.

      - formula **(Metric U Practice)**: Defines how the different *Metrics* and *Practices* are combined to computed the result.
- *GlobalResult:* This meta-class holds the global quality of a given SUS. The result is computed using the *Results* obtained from *Metrics* and *Practices* according to the specific *AssessmentMethod.*
    - *Attributes:*

      - interpretation **(String)**: Express how the result of the *AssessmentMethod* must be interpreted.

      - result **(Float)**: Holds the global quality value of a SUS according to an *AssessmentMethod.*

      - timestamp **(Date)**: Information regarding when the quality result has been computed.

      - version **(Float)**: Current version of the SUS on which the quality value has been computed.
- *Transformation:* The *Transformation* meta-class refers to a *TransformationUnit* from the *Transformation Meta-Model.* This *TransformationUnit* will manage all the necessary *TransformationUnits* (if more than one is required) and it will establish the order in which they must be triggered accordingly to the *Transformation Meta-Model.* Please,

refer to the *Transformation Meta-Model* section for more information about Transformation Units.

- *Artifact:* The *Artifact* meta-class refers to any element of the Software Development Life Cycle, such as code, classes of a model or the model itself. In this case, it is represented by the *Meta-ModelElement* from the Transformation Meta-Model. Please, refer to the *Transformation Meta-Model* section for more information about *Meta-ModelElements*.
- *ContextModel:* As a same Quality Criterion can have different quality interpretations regarding the context in which the interaction is taking place, the Quality Meta-Model needs to know exactly what the context is and how it is defined. Linking the Context Model to the Recommendation meta-class will allow to the quality experts to define different Recommendations regarding the different contexts in which the interaction can occur.

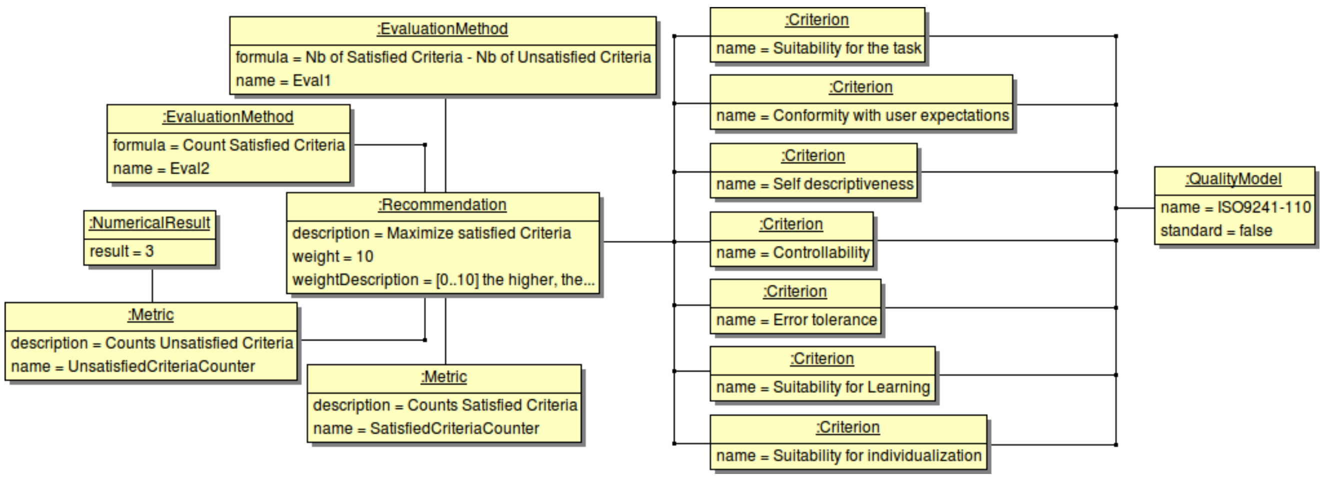

:EvaluationMethod
formula = Nb of Satisfied Criteria - Nb of Unsatisfied Criteria
name = Eval1
:EvaluationMethod
formula = Count Satisfied Criteria
name = Eval2
:NumericalResult
result = 3
:Recommendation
description = Maximize satisfied Criteria
weight = 10
weightDescription = [0..10] the higher, the...
:Metric
description = Counts Unsatisfied Criteria
name = UnsatisfiedCriteriaCounter
:Metric
description = Counts Satisfied Criteria
name = SatisfiedCriteriaCounter
:Criterion
name = Suitability for the task
:Criterion
name = Conformity with user expectations
:Criterion
name = Self descriptiveness
:Criterion
name = Controllability
:Criterion
name = Error tolerance
:Criterion
name = Suitability for Learning
:Criterion
name = Suitability for individualization
:QualityModel
name = ISO9241-110
standard = false

Figure 25 - Subset of the Quality Model of Ergonomic Criteria

3.9.7. How to build a Quality Model

We describe the steps that the quality expert must follow to define a Quality Model. This description involves identifying which classes of the Quality Meta-Model (Figure 22) are instantiated for each quality perspective.

1. Firstly, the quality expert must identify which quality standard is the more appropriate to fit the product requirements, i.e., identify the relevant elements in the specification of the SUS that are related to Quality. These requirements are the *Expected Quality.* Once the *Expected Quality* is extracted from the SUS specification, the quality expert can select the best Quality Model for such requirements (for instance an ISO standard, or any other standard such as a customized Quality Model developed by the quality expert).
2. The Quality Meta-Model can be instantiated now to represent the desired Quality Model for the particular product. For this, the *Criterion* meta-class is instantiated using the *CriterionAssociation meta-class* to structure the Criteria conforming to the selected Quality Model. An example is shown in the right side of figure 5. Once all the Criteria has been defined, specifying attributes and linking Criteria through the *CriterionAssociation* meta-class, the attribute *standard* from the Quality-Model meta-class is set to true. This indicates that only a standard is represented at this point and no other classes are instantiated yet (such as metrics or transformations). This allows the quality expert to re-use different quality models for other projects.

3. Thirdly, the quality expert can define the necessary recommendations based on different metrics and practices, as well as *AssessmentMethods* to allow the system to perform automatic quality evaluations. To do this, the quality expert will turn the *standard* attribute to false and will extend the Quality Model with all the necessary *Recommendations*, *Metrics*, *Practices* and *AssessmentMethods*. This new extended version of the Quality Model is able to compute the *Achieved Quality* through *AssessmentMethods*. For those *Practices* that cannot be automatically evaluated such as *Antipatterns*, the quality expert can express the necessary *LogicalResults (*for instance if an *Antipattern* is present or not*).*
4. The next step involves the definition of *Limits* of values for the desired metrics in case the system has some. This is done instantiating the *Limits* meta-class for each desired metric. This part of the Quality Model holds the *Wished Quality,* i.e., the values the metrics must ideally reach.
5. The last step consists in defining the *TransformationUnits* and *Meta-ModelElements* to increase the quality when the *Achieved Quality* is not enough.

Note that different iterations can be done in order to achieve the expected quality. For instance, if the result of a metric is not achieved, i.e., the value of the *NumericalResult* is not between the limit values, a transformation can be launched (if it has been specified) and performed on one or more Artifacts trying to achieve the desired value. Then, the global quality can be recalculated again and compared to the previous quality before transformation.

3.9.8. Example

The following example uses the QOC Meta-Model to evaluate the quality of an User Interface based on the Quality Meta-Model. Please, refer to the QOC Meta-Model section for more information regarding this meta-model. Figure 26 shows an instance of a QOC model in which designers propose several interactors to let the user enter a date. The first interactor is composed of three text fields for the day, month and year respectively, and a label indicating format notations. The second interactor is a calendar. We want to systematically evaluate what option is the best in terms of quality. For this, the quality expert choose the ISO 9241-110 standard as the Quality Model. Figure 27 shows an instantiation of this standard representing Ergonomic Criteria in HCI [9]. We explain only the three criteria that are relevant for this example:

- *Suitability for the task*: A dialog is suitable for the task if the dialog helps the user to complete her/his task in an effective and efficient manner.
- *Self descriptiveness*: A dialog is self descriptive if every single dialog step can immediately be understood by the user based on the information displayed by the system.
- *Error tolerance*: A dialog is fault tolerant if a task can be completed without erroneous inputs with minimal overhead for corrections by the human user.

Regarding the Figure 26, the first interactor does satisfy the three criteria whilst the text fields interactor does not. In this example, we are going to quantify this knowledge using the instance of the Quality Meta-Model depicted in Figure 25.

To quantify the quality of both alternatives, Figure 25, Figure 27 and Figure 28 link both the QOC Model and the Quality Model. The comparison between the two design options (Text fields versus calendar) is based on evaluation methods depicted in Figure 25. The *AssessmentMethod* instances (left part of Figure 25) use the following formulas for computation:

- Eval1 = Number of satisfied criteria - Number of unsatisfied criteria
- Eval2 = Number of satisfied criteria

The computation is based on the number of satisfied vs unsatisfied criteria in Figure 25:

- Eval1(Calendar) = 3 - 0 = 3

- Eval1(Text fields) = 3 - 3 = 0

and for the second evaluation method:

- Eval2(Calendar) = 3
- Eval2(Text fields) = 0

which concludes that the calendar option has a better quality than the text fields, accordingly to:

- The three criteria from the ISO 9241-110.
- The evaluation formulas Eval1 and Eval2

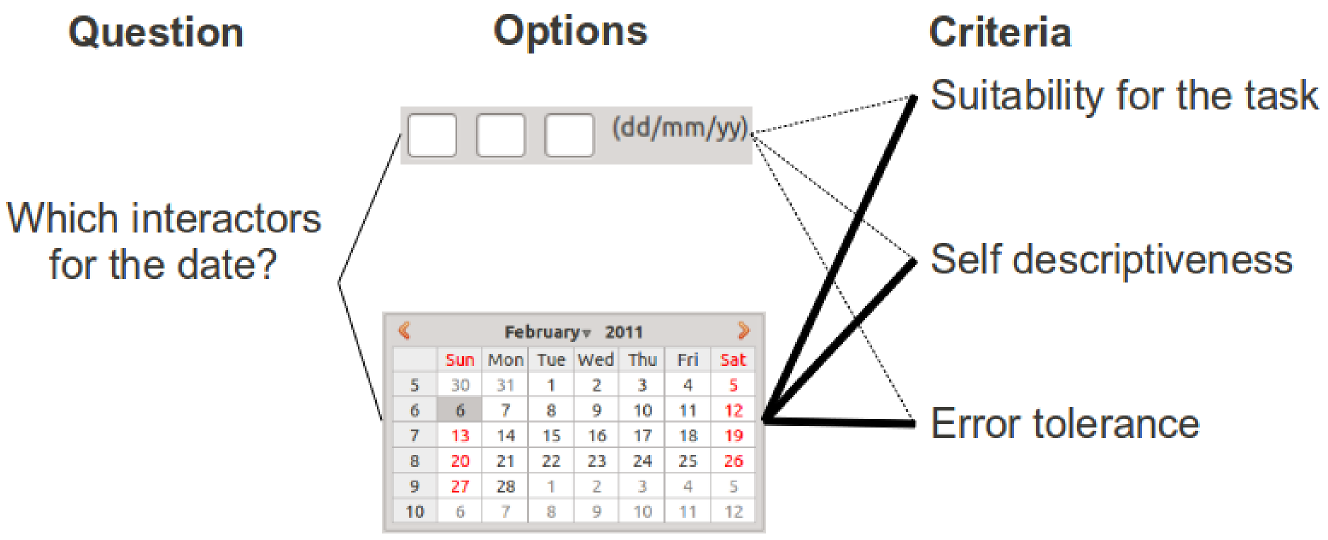


Figure 26 - QOC Model example for choosing interactors

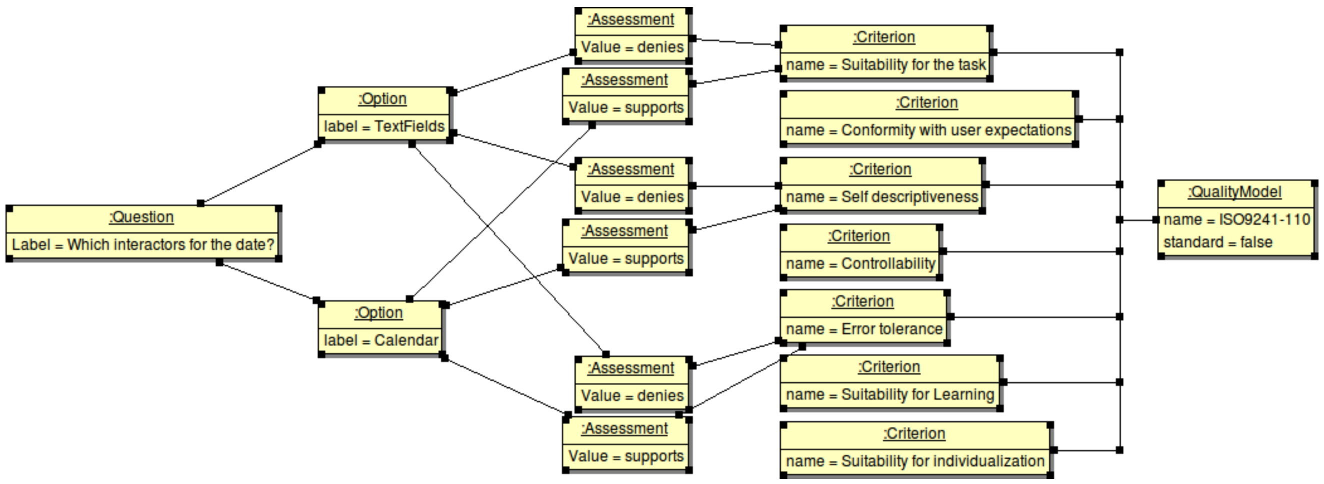


Figure 27 - . **Quality Model linked to the previous QOC Model**

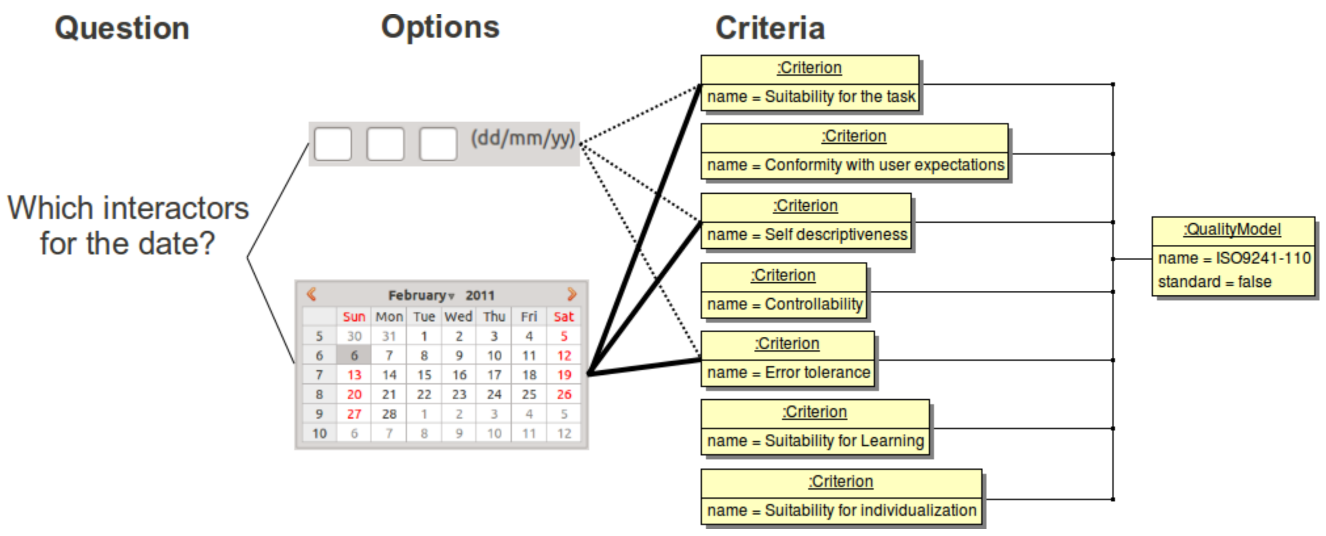
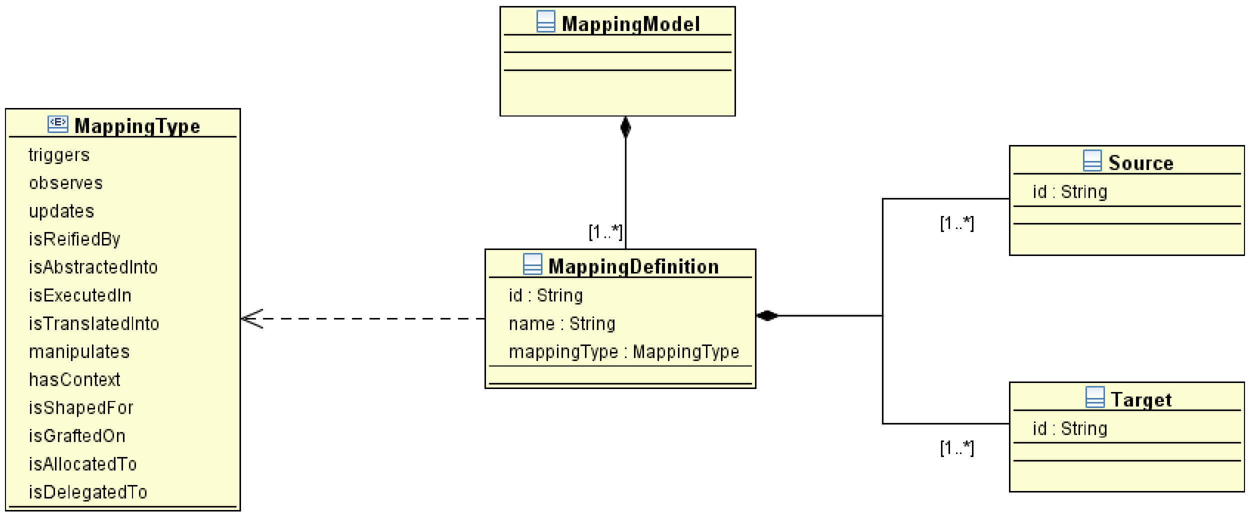

Question
Options
Criteria
Which interactors for the date?
(dd/mm/yy)
February 2011
Sun Mon Tue Wed Thu Fri Sat
:Criterion
name = Suitability for the task
:Criterion
name = Conformity with user expectations
:Criterion
name = Self descriptiveness
:Criterion
name = Controllability
:Criterion
name = Error tolerance
:Criterion
name = Suitability for Learning
:Criterion
name = Suitability for individualization
:QualityModel
name = ISO9241-110
standard = false

Figure 28 - Quality Model linked to the previous QOC Model (graphical version)

3.10. Mapping

The Mapping meta-model is the mapping aimed at describing the relationships allowing relating the different models.

3.10.1. Overview



- *MappingModel*: Is a model containing a serie of related mappings between models or elements of models. A mapping model serves to gather a set of inter-model relationships that are semantically related.
- *Source*: Designates one or several source(s) of a relationship whose it is part of.
  - *Attributes:*
    - id **(String)**: Identification string of the *Source*.
- *Target*: Designates one or several target(s) of a relationship whose it is part of.
  - *Attributes:*
    - id **(String)**: Identification string of the *Target*.
- *MappingDefinition*: Is any type of relationship established between one or many source models and one or many target models. A typical *MappingDefinition* is established between one source model and one target model, but it can be easily imagined that such a relationship can start from one source model to many target models, but from many source models to many target models. The mappings between the different models are of the types of the subtypes of the class *MappingDefinition*. It is expected to come up with a catalog of canonical relationships containing both the basic relationships and the one-to-many and many-to-many relationships expressed as a linear combination of the basic ones. Examples of such relationships are: - a task model is achieved in a dialog model - a task model is carried out by a user stereotype - a task model manipulates domain concepts - a task model is rendered through a particular interaction object - ... A *MappingDefinition* is the superclass of all possible relationships between models and elements of models. Consists of: one to many sources Consists of: one to many targets Constraint: the source should not necessarily be different from the target.
  - *Attributes:*
    - id **(String)**: Identification string of the *Source*.
    - name **(String)**: name of the *MappingDefinition*.
    - mappingType **(MappingType)**: type of the mapping relationship.
- *MappingType*:
  - *Attributes:*
    - triggers**:** Indicates a connection between a method of the domain model and a UI individual component (either at the abstract or at the concrete level).

- observes: Is a mapping between any UI component (at abstract or concrete level) and a domain attribute or instantiated attribute (at run time). Observes enables to specify that a UI component observes a value from the related domain concept.
- updates: Is a mapping between any UI component (at abstract or concrete level) and a domain attribute or instantiated attribute (at run time). Updates enables to specify that a UI component provides a value for the related domain concept.
- isReifiedBy: Is a model relationship involving a source model that is reified into a target model. This relationship is the inverse of *isAbstractedInto*. Maps the elements of an abstract user interface onto elements of a concrete user interface. In other words, this relationship specifies how any AbstractIU can be reified by a ConcreteIU. Constraint: the level of abstraction should be immediately superior to the one of the target model Constraint: the source model cannot be a model belonging to the Final User Interface.
- isAbstractedInto: Maps a concrete user interface element onto an abstract element.
- isExecutedIn: Indicates that a task is performed through one or several Abstract Compound IU and Abstract Elementary IU.
- isTranslatedInto: Enables to provide a trace of the adaptation of one component in another. IsAdaptedInto can be used while defining a transformation called translation.
- manipulates: Maps a task onto a domain concepts i.e., a class, an attribute, an operation or any combination of these types. This relationship has an attribute 'centrality' which specifies the relative importance of a domain concept to the execution of its corresponding task. This item is evaluate on a scale of 1 to 5. 1 meaning that the concept is not central, 5 that is completely central (i.e., essential to the execution of the task).
- hasContext: Relates any UI model or, in some cases, model elements to the context(s) for which it is supposed to apply.
- isShapedFor: Allows to associate a plasticity domain to a CUI.
- isGraftedOn: A task is grafted on another one.
- isAllocatedTo: A task is assigned to a resource. There are some allocation relationships for this assignment.
- isDelegatedTo: A resource who is assigned to a task allocates it to another resource. Example: The Chemist passed all of the work items allocated to him onto the Chemist Assistant.

4. CONCLUSION

This document has presented the last version of the meta models of UsiXML.

5. APPENDIX 1: MODIFICATIONS TO AUI METAMODEL

5.1. Executive summary

This appendix proposes a set of changes to the Abstract UI metamodel as was submitted to the W3C for standardization. The changes are motivated by several inconsistencies that have been found between the diagram that describes the metamodel, the XSD that was included in the document and the ontology definition that was included in the document.

Other changes are proposed because of best practices in the design of models and metamodels that can lead to a more easily understandable, verifiable and implementable metamodel.

5.2. Introduction

During the internal reviewing of the metamodels and the development and implementation of several tools on top of them a set of issues have risen that have made the implementation of the set of tools suboptimal when taking into account the goals of the project and the possibilities that the UsiXML metamodels should offer.

This appendix focuses on the AbstractUserInterface metamodel as it is defined in the submission that was sent to the W3C [W3C-SUB]. The changes are motivated by several issues that show inconsistencies between the three definitions that have been submitted of the the metamodel:

- The **diagram definition** of the metamodel shows the visual representation of the metamodel followed by the description of the elements and their properties as they appear in the diagram. Alas, this description of the elements is at times not clear enough, at times incomplete.
- The **XSD definition** of the metamodel leaves no space to ambiguity. Anyway there are some inconsistencies between the XSD definition and the diagram definition, such as missing properties and the representation of attributes as elements or sequences that complicate the design of the metamodel.
- The **ontology definition** of the metamodel describes many of the relationships that are not described in the XSD but that appear in the diagram representation. Anyway, in the ontology definition there are some relationships that are not correct or that are inconsistent with what is defined in the diagram representation and the XSD representation.

As part of the review and the implementation of tools that has been done we have elaborated a Change Proposal to correct the issues that we have found in the Abstract User Interface metamodel. This change proposal proposes a new XSD that is as similar as possible to the original metamodel but including changes so that:

- No new element is added, neither is any element removed from the metamodel.
- No new attribute or relationship is added neither is any attribute or relationship removed from the metamodel. If an attribute or relationship appears in the new XSD that is not in the XSD that was submitted to the W3C it is because of the homogenization of the XSD to include all the elements from the diagram representation and the ontology representation.
- The new XSD is consistent with the diagram representation and the ontology definition.
- The new XSD defines the attributes and the relationships of the elements in a consistent way.
- The new XSD allows for a better implementation of tools such as editors, validators, transformation, code generators, etc.

In the next sections the changes to the metamodel are explained in a detailed table, showing the changes from the current version of the XSD to the proposed XSD for each element of the metamodel.

5.3. Detailed changes

- **AbstractUIModel**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
relationships	compoundIUs [0..*]	compoundIUs [0..*]	compoundIUs [0..*] as sequence	abstractCompoundIUs [0..*] as sequence	compoundIUs [¿..?]

- **AbstractInteractionUnit**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
type			Defined as abstract	NOT defined as abstract	
relationships	parentIU [0..1] (inverse)	parentIU [0..1]	parentIU [0..1] as attribute	-	parentIU [¿..?]
	isNamed [0..*]	no_name [0..*]	isNamed [0..*] as sequence	name [0..*] as sequence	isNamed[¿..?]
	hasStates [0..*]	no_name [0..*]	hasStates [0..*] as sequence	state [0..*] as sequence	hasStates [¿..?]
	isListenBy [0..*]	no_name [0..*]	isListenBy [0..*] as sequence	listenter [0..*] as sequence	isListenBy [¿..?]

- **AbstractCompoundIU**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
relationships	belong [0..1] (inverse)	model [0..1]	belong [0..1] as attribute	-	belong [0..0]
	childIU [0..*]	interactionUnits [0..*]	childIU [0..*] as sequence	abstractInteractionUnit [0..*] as sequence	childIU [0..1]

- **AbstractElementaryIU**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
type			Defined as abstract	NOT defined as abstract	

- **AbstractSelectionIU**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes			orderCriteria defined as attribute	orderCriteria defined as sequence	

- **AbstractDataIU**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes			domainReference defined as attribute	domainReference defined as sequence	
			defaultValue defined as attribute	defaultValue defined as sequence	
			dataFormat defined as attribute	dataFormat defined as sequence	

- **AbstractTriggerIU**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes			triggerIUType defined as attribute	triggerIUType defined as sequence	

- **AbstractLocalization**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes	–	–	–	id as attribute	id : String
			label defined as attribute	label defined as sequence	
			longLabel defined as attribute	longLabel defined as sequence	
			shortLabel defined as attribute	shortLabel defined as sequence	
			descLabel defined as attribute	descLabel defined as sequence	
			help defined as attribute	help defined as sequence	

relationships	nameOf [0..1] (inverse)	-	nameOf [0..1] as attribute	-	nameOf [1..1]

- **AbstractListener**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes			name defined as attribute	name defined as sequence	
relationships	listen [0..1] (inverse)	no_name [1..1]	listen [0..1] as attribute	-	listen [1..1]
	isTriggerOn [1..*]	no_name [1..*]	isTriggerOn [1..*] as sequence	rules [1..*] as sequence	isTriggerOn [1..?]

- **Rule**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes	-	-	-	id as attribute	id : String
			name defined as attribute	name defined as sequence	
relationships	theAction [1..1]	no_name [1..1]	theAction [1..1] as sequence	action [1..1] as sequence	theAction [1..1]
	onEvent [1..1]	no_name [1..1]	onEvent [1..1] as sequence	event [1..1] as sequence	onEvent [1..1]
	isConditionnedBy [0..1]	no_name [0..1]	isConditionnedBy as sequence	condition [0..1] as sequence	isConditionnedBy [0..1]
	isJustifiedBy [0..1]	no_name [0..1]	isJustifiedBy [0..1] as sequence	justification [0..1] as sequence	isJustifiedBy [0..1]
	trigger [0..1] (inverse)	no_name [1..1]	trigger [0..1] as attribute	-	trigger [0..1]

- **Condition**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes	-	-	-	id as attribute	id : String
			specification defined as attribute	specification defined as sequence	

relationships	conditionOf [0..1] (inverse)	no_name [1..1]	conditionOf [0..1] as attribute	-	conditionOf [1..1]

- **Justification**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes	-	-	-	id as attribute	id : String
			content defined as attribute	content defined as sequence	
			type defined as attribute	type defined as sequence	
relationships	justify [0..1] (inverse)	-	justify [0..1] as attribute	-	justify [1..1]

- **ActionExpression**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
type			Defined as abstract	NOT defined as abstract	
attributes			name defined as attribute	name defined as sequence	
relationships	actionOfRule [0..1] (inverse)	-	actionOfRule [0..1] as attribute	-	actionOf [1..1]
	actionOfTransition [0..1] (inverse)	-	actionOfTransition [0..1] as attribute	-	actionOf [1..1]

- **EventExpression**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
type			Defined as abstract	NOT defined as abstract	
attributes			name defined as attribute	name defined as sequence	
relationships	eventOfRule [0..1] (inverse)	-	eventOfRule [0..1] as attribute	-	eventOf [1..1]
	eventOfTransition [0..1] (inverse)	-	eventOfTransition [0..1] as attribute	-	eventOf [1..1]
	eventOfPrevious	-	eventOfPrevious	-	-

	[0..1] (inverse)		[0..1] as attribute		
	eventOfNext [0..1] (inverse)	-	eventOfNext [0..1] as attribute	-	-

- **AtomicAction**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes			type defined as attribute	type defined as sequence	
			specification defined as attribute	specification defined as sequence	

- **TemporalActionExpression**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes			type defined as attribute	type defined as sequence	

- **AtomicEvent**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes			type defined as attribute	type defined as sequence	
			specification defined as attribute	specification defined as sequence	

- **TemporalEventExpression**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes			type defined as attribute	type defined as sequence	
relationships	next [0..1] (inverse)	no_name [1..1]	next [0..1] as sequence	next [1..1] as sequence	next [1..1]
	previous [0..1] (inverse)	no_name [1..1]	previous [0..1] as sequence	previous [1..1] as sequence	previous [1..1]

- **State**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes			description defined as attribute	description defined as sequence	
relationships	stateIU [0..1] (inverse)	-	stateIU [0..1] as attribute	-	stateIU [¿..?]
	source [1..*]	source [¿..?] (not defined as composition)	source [1..*] as sequence	source [1..*] as sequence	source [1..1]
	target [0..*] (inverse)	-	target [0..*] as attribute	target [0..*] as sequence	target [0..*]

- **Transition**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
attributes	-	-	-	transitionExpression defined as sequence	transitionExpression : String
relationships	sourceOf [0..1] (inverse)	-	sourceOf [0..1] as attribute	-	sourceOf [1..1]
	targetOf [0..1]	no_name [¿..?]	targetOf [0..1] as sequence	-	targetOf [1..1]
	target [0..*] (inverse)	-	target [0..*] as attribute	target [0..*] as sequence	target [0..*]
	thenAction [1..1] (inverse)	no_name [1..1]	thenAction [0..*] as sequence	action [1..1] as sequence	thenAction [1..1]
	onEvent [1..1] (inverse)	no_name [1..1]	onEvent [0..*] as sequence	event [1..1] as sequence	onEvent [1..1]

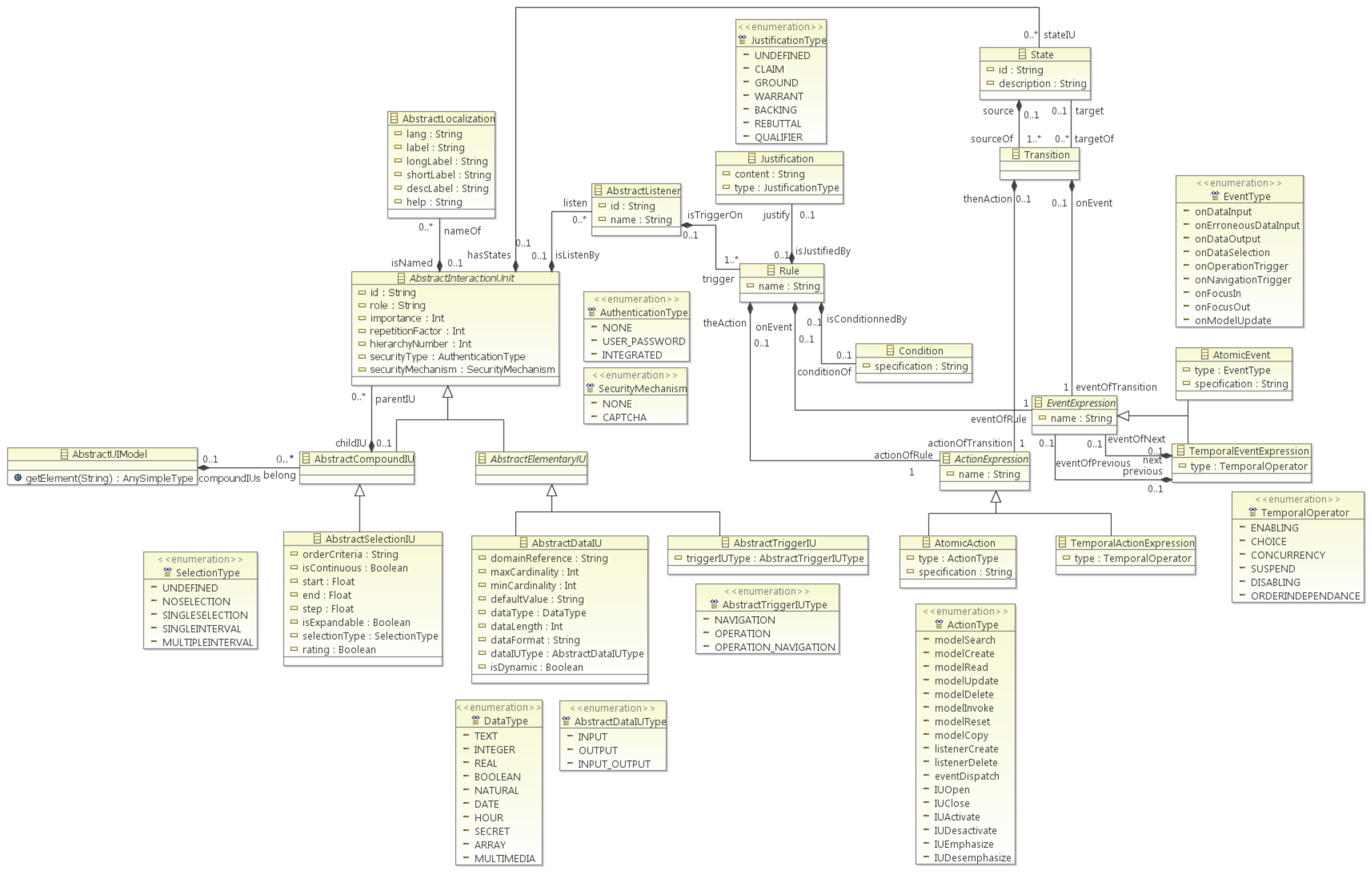
- **DataType**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
type			DataType	DataTypeType	

- **EventType**

	New Metamodel	Old Metamodel	New XSD	Old XSD	Ontologies
literal	`onFocusIn`	`onFocusIn (in the diagram) and onIUFocusIn (in the description)`			
	`onFocusOut`	`onFocusOut (in the diagram) and onIUFocusOut (in the description)`			

5.4. The final AUI Metamodel



WP Leader / Task Leader	THALES INTERNAL DOCUMENT NUMBER	PAGE
UCL / UCL	61 566 104/179/13	110/121
	Revision	**1**

5.5. Proposed Abstract User Interface XSD Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns:aui="http:///usixml.aui_0.1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http:///usixml.aui_0.1.0">
    <xsd:element name="AbstractCompoundIU" type="aui:AbstractCompoundIU" />
    <xsd:element name="AbstractDataIU" type="aui:AbstractDataIU" />
    <xsd:element name="AbstractElementaryIU" type="aui:AbstractElementaryIU" />
    <xsd:element name="AbstractInteractionUnit" type="aui:AbstractInteractionUnit"
/>
    <xsd:element name="AbstractListener" type="aui:AbstractListener" />
    <xsd:element name="AbstractLocalization" type="aui:AbstractLocalization" />
    <xsd:element name="AbstractSelectionIU" type="aui:AbstractSelectionIU" />
    <xsd:element name="AbstractTriggerIU" type="aui:AbstractTriggerIU" />
    <xsd:element name="AbstractUIModel" type="aui:AbstractUIModel" />
    <xsd:element name="ActionExpression" type="aui:ActionExpression" />
    <xsd:element name="AtomicAction" type="aui:AtomicAction" />
    <xsd:element name="AtomicEvent" type="aui:AtomicEvent" />
    <xsd:element name="Condition" type="aui:Condition" />
    <xsd:element name="EventExpression" type="aui:EventExpression" />
    <xsd:element name="Justification" type="aui:Justification" />
    <xsd:element name="Rule" type="aui:Rule" />
    <xsd:element name="State" type="aui:State" />
    <xsd:element name="TemporalActionExpression"
type="aui:TemporalActionExpression" />
    <xsd:element name="TemporalEventExpression" type="aui:TemporalEventExpression"
/>
    <xsd:element name="Transition" type="aui:Transition" />
    <xsd:complexType name="AbstractCompoundIU">
        <xsd:complexContent>
            <xsd:extension base="aui:AbstractInteractionUnit">
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
name="childIU"
                        type="aui:AbstractInteractionUnit" />
                </xsd:sequence>
                <xsd:attribute name="belong" type="xsd:anyURI" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="AbstractDataIU">
        <xsd:complexContent>
            <xsd:extension base="aui:AbstractElementaryIU">
                <xsd:attribute name="domainReference" type="xsd:string" />
                <xsd:attribute default="-1" name="maxCardinality"
                    type="xsd:int" />
                <xsd:attribute default="0" name="minCardinality" type="xsd:int" />
                <xsd:attribute name="defaultValue" type="xsd:string" />
                <xsd:attribute name="dataType" type="aui:DataType" />
                <xsd:attribute default="-1" name="dataLength" type="xsd:int" />
                <xsd:attribute name="dataFormat" type="xsd:string" />
                <xsd:attribute name="dataIUType" type="aui:AbstractDataIUType" />
                <xsd:attribute default="false" name="isDynamic" type="xsd:boolean"
/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
```

```
<xsd:complexType abstract="true" name="AbstractElementaryIU">
    <xsd:complexContent>
        <xsd:extension base="aui:AbstractInteractionUnit" />
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType abstract="true" name="AbstractInteractionUnit">
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="isListenBy"
            type="aui:AbstractListener" />
        <xsd:element maxOccurs="unbounded" minOccurs="0"
            name="isNamed" type="aui:AbstractLocalization" />
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="hasStates"
            type="aui:State" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" />
    <xsd:attribute default="" name="role" type="xsd:string" />
    <xsd:attribute default="3" name="importance" type="xsd:int" />
    <xsd:attribute name="repetitionFactor" type="xsd:int" />
    <xsd:attribute name="hierarchyNumber" type="xsd:int" />
    <xsd:attribute name="securityType" type="aui:AuthenticationType" />
    <xsd:attribute name="securityMechanism" type="aui:SecurityMechanism" />
    <xsd:attribute name="parentIU" type="xsd:anyURI" />
</xsd:complexType>
<xsd:complexType name="AbstractListener">
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" name="isTriggerOn"
            type="aui:Rule" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" />
    <xsd:attribute name="name" type="xsd:string" />
    <xsd:attribute name="listen" type="xsd:anyURI" />
</xsd:complexType>
<xsd:complexType name="AbstractLocalization">
    <xsd:attribute default="" name="lang" type="xsd:string" />
    <xsd:attribute default="" name="label" type="xsd:string" />
    <xsd:attribute default="" name="longLabel" type="xsd:string" />
    <xsd:attribute default="" name="shortLabel" type="xsd:string" />
    <xsd:attribute default="" name="descLabel" type="xsd:string" />
    <xsd:attribute default="" name="help" type="xsd:string" />
    <xsd:attribute name="nameOf" type="xsd:anyURI" />
</xsd:complexType>
<xsd:complexType name="AbstractSelectionIU">
    <xsd:complexContent>
        <xsd:extension base="aui:AbstractCompoundIU">
            <xsd:attribute default="alphabetically" name="orderCriteria"
                type="xsd:string" />
            <xsd:attribute name="isContinuous" type="xsd:boolean" />
            <xsd:attribute default="1" name="start" type="xsd:float" />
            <xsd:attribute default="20" name="end" type="xsd:float" />
            <xsd:attribute default="5" name="step" type="xsd:float" />
            <xsd:attribute default="false" name="isExpandable"
                type="xsd:boolean" />
            <xsd:attribute name="selectionType" type="aui:SelectionType" />
            <xsd:attribute default="false" name="rating" type="xsd:boolean" />
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="AbstractTriggerIU">
    <xsd:complexContent>
```

```
            <xsd:extension base="aui:AbstractElementaryIU">
                <xsd:attribute name="triggerIUType"
type="aui:AbstractTriggerIUType" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="AbstractUIModel">
        <xsd:annotation>
            <xsd:appinfo>
                <operation name="getElement" type="xsd:anySimpleType">
                    <parameter name="reference" type="xsd:string" />
                </operation>
            </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0"
                name="compoundIUs" type="aui:AbstractCompoundIU" />
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType abstract="true" name="ActionExpression">
        <xsd:attribute name="name" type="xsd:string" />
        <xsd:attribute name="actionOfRule" type="xsd:anyURI" />
        <xsd:attribute name="actionOfTransition" type="xsd:anyURI" />
    </xsd:complexType>
    <xsd:complexType name="AtomicAction">
        <xsd:complexContent>
            <xsd:extension base="aui:ActionExpression">
                <xsd:attribute name="type" type="aui:ActionType" />
                <xsd:attribute name="specification" type="xsd:string" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="AtomicEvent">
        <xsd:complexContent>
            <xsd:extension base="aui:EventExpression">
                <xsd:attribute name="type" type="aui:EventType" />
                <xsd:attribute name="specification" type="xsd:string" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Condition">
        <xsd:attribute name="specification" type="xsd:string" />
        <xsd:attribute name="conditionOf" type="xsd:anyURI" />
    </xsd:complexType>
    <xsd:complexType abstract="true" name="EventExpression">
        <xsd:attribute name="name" type="xsd:string" />
        <xsd:attribute name="eventOfRule" type="xsd:anyURI" />
        <xsd:attribute name="eventOfTransition" type="xsd:anyURI" />
        <xsd:attribute name="eventOfNext" type="xsd:anyURI" />
        <xsd:attribute name="eventOfPrevious" type="xsd:anyURI" />
    </xsd:complexType>
    <xsd:complexType name="Justification">
        <xsd:attribute name="content" type="xsd:string" />
        <xsd:attribute name="type" type="aui:JustificationType" />
        <xsd:attribute name="justify" type="xsd:anyURI" />
    </xsd:complexType>
    <xsd:complexType name="Rule">
        <xsd:sequence>
```

```
            <xsd:element minOccurs="0" name="isJustifiedBy"
type="aui:Justification" />
            <xsd:element name="theAction" type="aui:ActionExpression" />
            <xsd:element minOccurs="0" name="isConditionnedBy"
type="aui:Condition" />
            <xsd:element name="onEvent" type="aui:EventExpression" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" />
        <xsd:attribute name="trigger" type="xsd:anyURI" />
    </xsd:complexType>
    <xsd:complexType name="State">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="source"
                type="aui:Transition" />
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:string" />
        <xsd:attribute name="description" type="xsd:string" />
        <xsd:attribute name="stateIU" type="xsd:anyURI" />
        <xsd:attribute name="target">
            <xsd:simpleType>
                <xsd:list itemType="xsd:anyURI" />
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
    <xsd:complexType name="TemporalActionExpression">
        <xsd:complexContent>
            <xsd:extension base="aui:ActionExpression">
                <xsd:attribute name="type" type="aui:TemporalOperator" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="TemporalEventExpression">
        <xsd:complexContent>
            <xsd:extension base="aui:EventExpression">
                <xsd:sequence>
                    <xsd:element minOccurs="0" name="next"
type="aui:EventExpression" />
                    <xsd:element minOccurs="0" name="previous"
type="aui:EventExpression" />
                </xsd:sequence>
                <xsd:attribute name="type" type="aui:TemporalOperator" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Transition">
        <xsd:sequence>
            <xsd:element name="thenAction" type="aui:ActionExpression" />
            <xsd:element name="onEvent" type="aui:EventExpression" />
        </xsd:sequence>
        <xsd:attribute name="sourceOf" type="xsd:anyURI" />
        <xsd:attribute name="targetOf" type="xsd:anyURI" />
    </xsd:complexType>
    <xsd:simpleType name="JustificationType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="UNDEFINED" />
            <xsd:enumeration value="CLAIM" />
            <xsd:enumeration value="GROUND" />
            <xsd:enumeration value="WARRANT" />
            <xsd:enumeration value="BACKING" />
            <xsd:enumeration value="REBUTTAL" />
            <xsd:enumeration value="QUALIFIER" />
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="AbstractTriggerIUType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="NAVIGATION" />
            <xsd:enumeration value="OPERATION" />
            <xsd:enumeration value="OPERATION_NAVIGATION" />
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="AbstractDataIUType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="INPUT" />
            <xsd:enumeration value="OUTPUT" />
            <xsd:enumeration value="INPUT_OUTPUT" />
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ActionType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="modelSearch" />
            <xsd:enumeration value="modelCreate" />
            <xsd:enumeration value="modelRead" />
            <xsd:enumeration value="modelUpdate" />
            <xsd:enumeration value="modelDelete" />
            <xsd:enumeration value="modelInvoke" />
            <xsd:enumeration value="modelReset" />
            <xsd:enumeration value="modelCopy" />
            <xsd:enumeration value="listenerCreate" />
            <xsd:enumeration value="listenerDelete" />
            <xsd:enumeration value="eventDispatch" />
            <xsd:enumeration value="IUOpen" />
            <xsd:enumeration value="IUClose" />
            <xsd:enumeration value="IUActivate" />
            <xsd:enumeration value="IUDesactivate" />
            <xsd:enumeration value="IUEmphasize" />
            <xsd:enumeration value="IUDesemphasize" />
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="AuthenticationType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="NONE" />
            <xsd:enumeration value="USER_PASSWORD" />
            <xsd:enumeration value="INTEGRATED" />
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="DataType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="TEXT" />
            <xsd:enumeration value="INTEGER" />
            <xsd:enumeration value="REAL" />
            <xsd:enumeration value="BOOLEAN" />
            <xsd:enumeration value="NATURAL" />
            <xsd:enumeration value="DATE" />
            <xsd:enumeration value="HOUR" />
            <xsd:enumeration value="SECRET" />
            <xsd:enumeration value="ARRAY" />
            <xsd:enumeration value="MULTIMEDIA" />
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="EventType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="onDataInput" />
            <xsd:enumeration value="onErroneousDataInput" />
            <xsd:enumeration value="onDataOutput" />
            <xsd:enumeration value="onDataSelection" />
            <xsd:enumeration value="onOperationTrigger" />
            <xsd:enumeration value="onNavigationTrigger" />
            <xsd:enumeration value="onFocusIn" />
            <xsd:enumeration value="onFocusOut" />
            <xsd:enumeration value="onModelUpdate" />
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="SecurityMechanism">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="NONE" />
            <xsd:enumeration value="CAPTCHA" />
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="SelectionType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="UNDEFINED" />
            <xsd:enumeration value="NOSELECTION" />
            <xsd:enumeration value="SINGLESELECTION" />
            <xsd:enumeration value="SINGLEINTERVAL" />
            <xsd:enumeration value="MULTIPLEINTERVAL" />
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="TemporalOperator">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="ENABLING" />
            <xsd:enumeration value="CHOICE" />
            <xsd:enumeration value="CONCURRENCY" />
            <xsd:enumeration value="SUSPEND" />
            <xsd:enumeration value="DISABLING" />
            <xsd:enumeration value="ORDERINDEPENDANCE" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

5.6. Conclusion

The changes presented in this appendix propose some modifications to the XSD of the Abstract User Interface metamodel as it was presented for submission to the W3C. The changes are motivated by several issues raised during review of the metamodels and during the implementation of tools. The changes aim at making the metamodel as homogeneous as possible without going much further, that is, not adding new elements or properties that we not defined in the first place.

We aim at getting the same flexibility with the new metamodel than with the existing one and making any existing tools be able to adapt to the changes with a minimum of effort. Also, the new metamodel would allow a more complete and straightforward implementation of tools, along with a wider field of compatibility among existing tools.

It is important to note that the proposed changes in the metamodel make the implementation of tools on top of the metamodeling framework of Eclipse [EMF] much more feasible and with a better integration between the EMF technology and the UsiXML technology.

REFERENCES

[A03]

Agrawal, A., "Graph Rewriting And Transformation (GReAT): A Solution For The Model Integrated Computing (MIC) Bottleneck," ase, pp.364, 18th IEEE International Conference on Automated Software Engineering (ASE'03), 2003

[B95]

Bramwell, C. Formal Development Methods for Interactive System: Combining Interactors and Design Rationale. Ph.D. Thesis. University of York. 1995.

[BBKLMM78]

Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., and Merritt, M. Characteristics of Software Quality, North Holland, 1978.

[C02]

Carlier A. Management de la qualité pour la maîtrise du SI, Paris, Hermès, p. 28, 2006.

[CCB02]

Calvary, G., Coutaz, J., Bouillon, L., Florins, M., Limbourg, Q., Marucci, L., Paternò, F., Santoro, C., Souchon, N., Thevenin, D., Vanderdonckt, J., 2002 *The CAMELEON Reference Framework*, Deliverable 1.1, CAMELEON Project

[**CCTLBV03]**

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

[CR91]

Carroll, J. M. and Rosson, M. B. Getting around the task-artifact cycle: How to make claims and design by scenario. ACM Trans. Inf. Syst. 10, 2 (Apr.), p 181–212, 1991.

[D96]

Dromey, R.G. Concerning the Chimera. IEEE Software 13 (1), p 33- 43, 1996.

[G10]

García Frey, A. Self-explanatory user interfaces by model-driven engineering. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems* (EICS '10). ACM, New York, NY, USA, p 341-344, 2010.

[ISO25010]

ISO/IEC CD 25010.3: Systems and software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Software product quality and system quality in use models. ISO, 2009.

[ISO9126]

ISO/IEC 9126-1: Software engineering. Product quality - Part 1: Quality model. ISO, 2001.

[ISO9241]

ISO 9241-110:Ergonomics of human-system interaction - Part 110: Dialogue principles. ISO, 2006.

[Jou05]

Frédéric Jouault, Ivan Kurtev. Transforming Models with ATL, presented at MODELS, 2005., vol 3844/2006, pages 128-138

[KWB03]

Anneke G. Kleppe, Jos Warmer, and Wim Bast. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[KSC09]

Kashif M, Si-Saïd Cherfi S., Comyn-Wattiau I. Data Quality Through Conceptual Model Quality – Reconciling Researchers and Practitioners Through a Customizable Quality Model. In International Conference on Information Quality (ICIQ), 2009.

[LMR09]

López-Jaquero, V., Montero, F., Real, F. Designing User Interface Adaptation Rules with T:XML. In Proceedings of the 14th international Conference on intelligent User interfaces (Sanibel Island, USA, February 08-11, 2009). IUI '09. ACM, New York, NY.

[LP07]

Xavier Lacaze and Philippe A. Palanque. DREAM & TEAM: A Tool and a Notation Supporting Exploration of Options and Traceability of Choices for Safety Critical Interactive Systems. In Maria Cecilia Calani Baranauskas, Philippe A. Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa, editors, Human-Computer Interaction - INTERACT 2007, 11th IFIP TC 13 International Conference, Rio de Janeiro, Brazil, September 10-14, 2007, Proceedings, Part II, volume 4662 of Lecture Notes in Computer Science, pages 525–540. Springer-Verlag, Berlin, 2007.

[LPB+07]

Xavier Lacaze, Philippe Palanque, Eric Barboni, Rémi Bastide, and David Navarre. From DREAM to Realitiy: Specificities of Interactive Systems Development with respect to Rationale Management. In Allen H. Dutoit, Raymond McCall, Ivan Mistrik, and Barbara Paech, editors, Rationale Management in Software Engineering, pages 155–172. Springer-Verlag, 2007.

[LVM+04]

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López Jaquero, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces, 9th IFIP Working Conf. on Engineering for Human-Computer Interaction. EHCI-DSVIS'2004, Springer, 2005, pp. 200-220.

[M91]

McCall, R. J. PHI: A conceptual foundation for design hypermedia. Des. Stud. 12, 1, p 30 – 41, 1991.

[MC96]

Moran, T. P. and Carroll, J. M. Overview of design rationale. In Design Rationale: Concepts, Techniques, and Use, T. P. Moran and J. M. Carroll, Eds. LEA computers, cognition, and work series. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, p 1–19, 1996.

[MCG04]

Tom Mens, Krzysztof Czarnecki, and Pieter Van Gorp. 04101 Discussion - A Taxonomy of Model Transformations. In Jean Bézivin and Reiko Heckel, editors, Language Engineering for Model-Driven Software Development, volume 04101 of Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2004.

[MD96]

Mohagheghi, P. and Dehlen, V. A Metamodel for Specifying Quality Models in Model-Driven Engineering. Nordic Workshop on Model Driven Engineering NW-MoDE '08, Reykjavik Iceland, p 20-22, August 2008.

[MLNPW10]

Martinie De Almeida, C., Ladry, J.F., Navarre D., Palanque P., Winckler, M. A. Embedding Requirements in Design Rationale to Deal Explicitly with User eXperience and Usability in an "intensive" Model-Based

Development Approach (regular paper). In: Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI 2010), Atlanta Georgia USA, Vol. 617, (Eds.), CEUR Workshop Proceedings, p. 29-32, 2010.

[MRW77]

McCall, J. A., Richards, P. K., and Walters, G. F. Factors in Software Quality, Nat'l Tech. Information Service, no. Vol. 1, 2 and 3, 1977.

[MYBM96]

Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, and Thomas P. Moran. Questions, options, and criteria: elements of design space analysis. pages 53–105, 1996.

[N05]

Nogier, J.F. De l'ergonomie du logiciel au design des sites Web, Third edition, Dunod 2005.

[OMG08]

Object Management Group, (2007), Unified Modeling Language 2.0, in formal/2007-02-03, 2007.

[OMG09]

OMG, "Unified Modeling Language: Superstructure", version 2.2, February 2009. OMG Document Number: formal/2009-02-02

[OMG10]

Object Management Group, (2010), Business Process Modeling Notation 2.0, in formal Document -- dtc/10-06-04

[PL07]

Palanque P. and Lacaze, X. DREAM-TEAM: A Tool and a Notation Supporting Exploration of Options and Traceability of Choices for Safety Critical Interactive Systems. In Proceedings of INTERACT 2007, Rio, Brazil, Lecture Notes in Computer Science, Springer Verlag, September 2007.

[SCF05]

Jean-Sebastien Sottet, Gaëlle Calvary, and Jean-Marie Favre. Towards Model-Driven Engineering of Plastic User Interfaces. In Andreas Pleuß, Jan Van den Bergh, Heinrich Hußmann, and Stefan Sauer, editors, MDDAUI '05, Model Driven Development of Advanced User Interfaces 2005, Proceedings of the MoDELS'05 Workshop on Model Driven Development of Advanced User Interfaces, Montego Bay, Jamaica, October 2, 2005, volume 159 of CEUR Workshop Proceedings. CEUR-WS.org, 2005.

[SAC02]

Si-Saïd Cherfi S., Akoka J., Comyn-Wattiau I. Conceptual Modeling Quality - From EER to UML Schemas Evaluation, Lecture Notes in Computer Science, Vol. 2503, p 414-428, January 2002.

[SDKP06]

Seffah, A., Donyaee, M., Kline, R. and Padda, H. Usability measurement and metrics: A consolidated model. Software Quality Journal, 14(2), p 159–178, June 2006.

[Seo06]

Hong-Seok Na, O-Hoon Choi, Jung-Eun Lim. A Metamodel-Based Approach for Extracting Ontological Semantics from UML Models. WEB INFORMATION SYSTEMS – WISE 2006: 411-422, Volume 4255/2006.

[Sta08]

Adrian Stanciulescu. A Methodology for Developing Multimodal User Interfaces of Information Systems. PhD thesis, Université catholique de Louvain, June 2008.

[T00]

Taentzer, G. 2000. AGG: A Tool Environment for Algebraic Graph Transformation. In Proc. of the Int. Workshop on Applications of Graph Transformations with industrial Relevance. LNCS, vol. 1779. Springer, London, 481-488.

[UsiXML07]

UCL, (2007), UsiXML V1.8 Reference Manual, February 2007.