



Results from the Investigation Teams on: Declarative UI Authoring & Context Model and Universal APIs

Jean Vanderdonckt, José Manuel Cantera Fonseca
Jose Luis Diaz Diaz, Juan Manuel Gonzalez Calleros

Partners:
Université catholique de Louvain (UCL), Telefónica I+D (TID)
ISTI-CNR, UPM, Nokia Siemens Networks, UIB

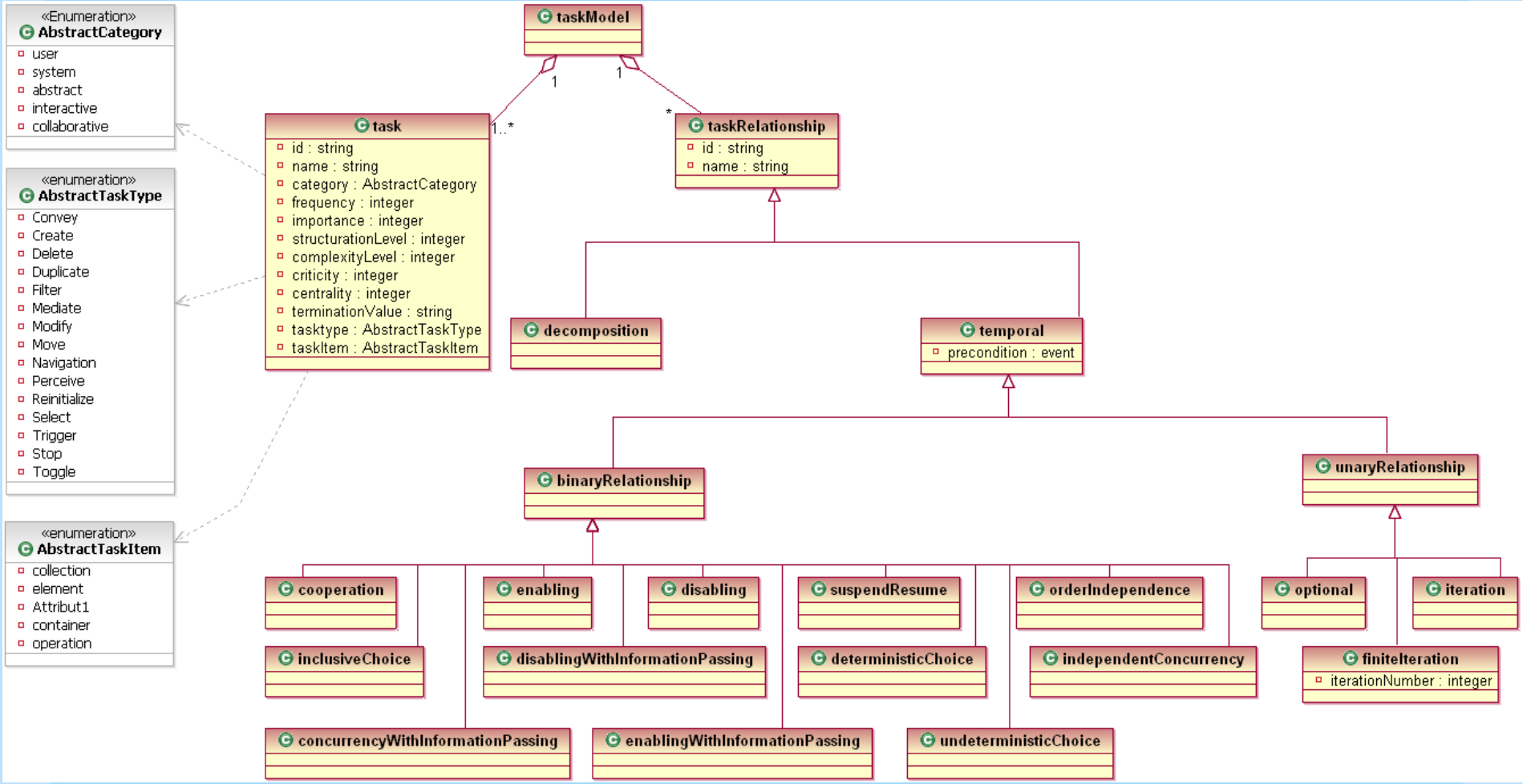
USIXML

Process

- This is the result of the following process:
  - Kick-off meeting in Brussels October 21-22 2008
  - Regular teleconferences with involved partners
  - Email discussion list
  - Two face to face meetings
    - Louvain-la-Neuve January 29-30
    - Louvain-la-Neuve April 23-24
  - Sanity check on existing works:
    - TeresaXML
    - EMode
    - Xforms
    - UsiXML
    - MASP project

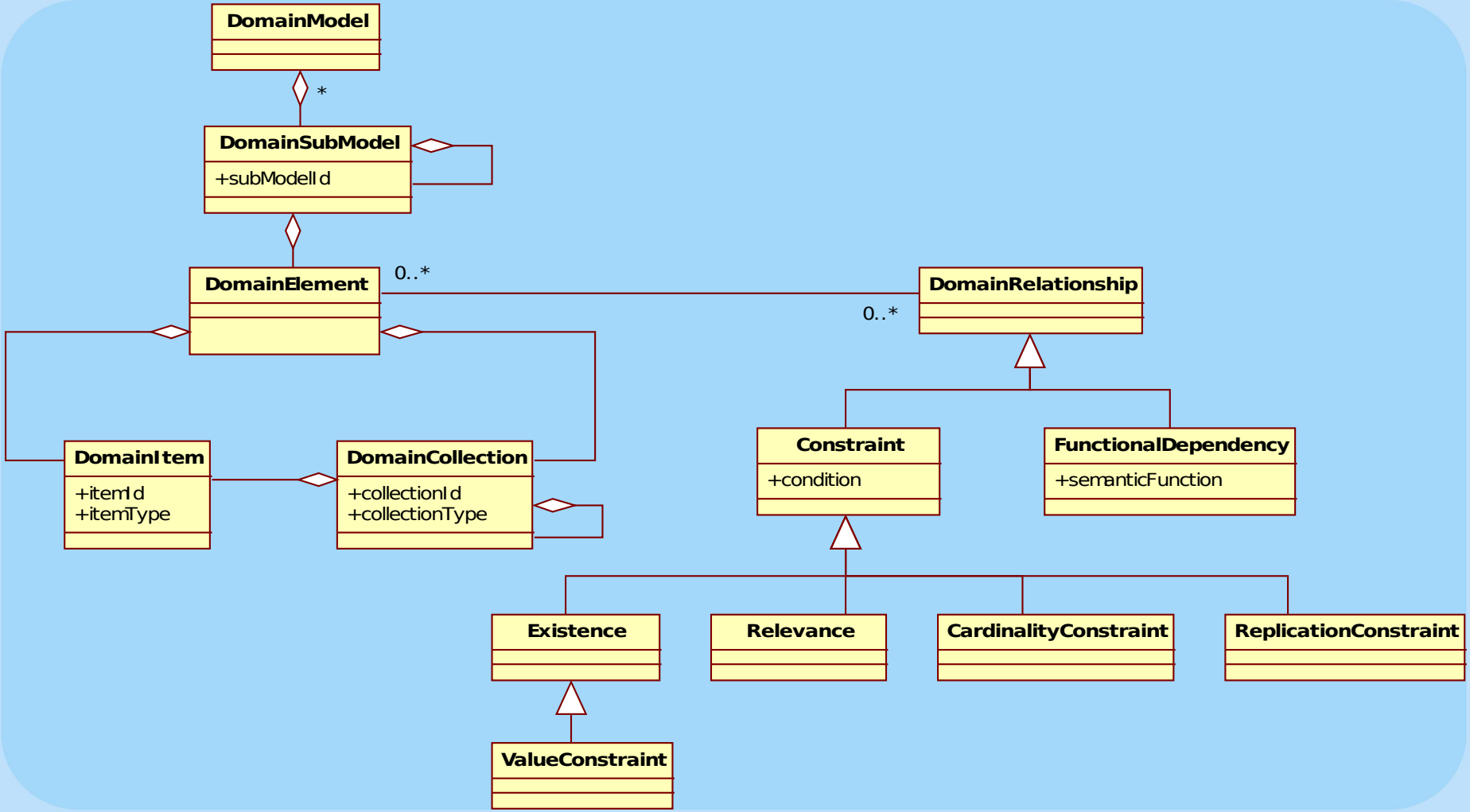
Application task models, data and meta-data

Task Model



USIXML

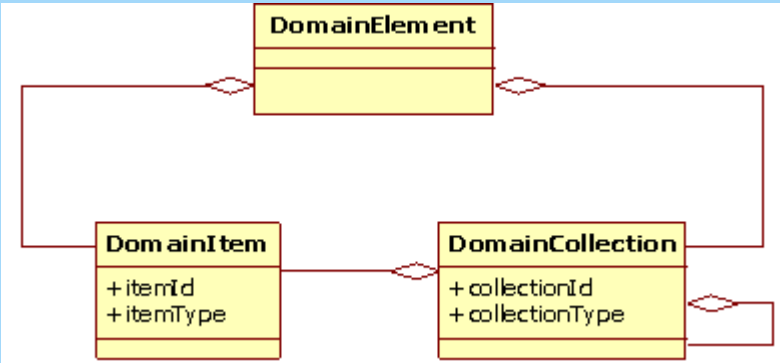
Domain Model



Domain Model

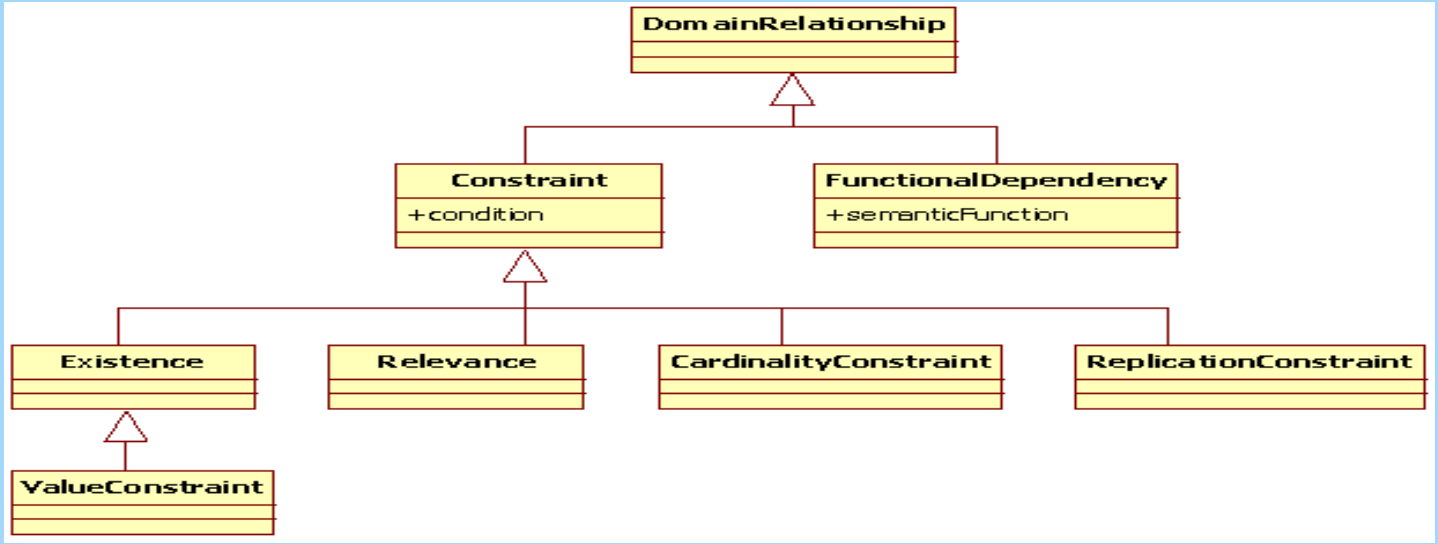
- The *Domain model* is a description of the classes of objects manipulated by a user while interacting with a system.
- *DomainSubModel,* which is a way to represent a domain model that at the same time can be composed of a different domain sub-model. For instance, Java packages where different submodels can be within a model and they can be associated to different domain facets.
- A *DomainElement* describes the characteristics of a set of objects sharing a set of common properties.
- The *DomainRelationship* describes various types of relationships between domain elements.

Domain Model



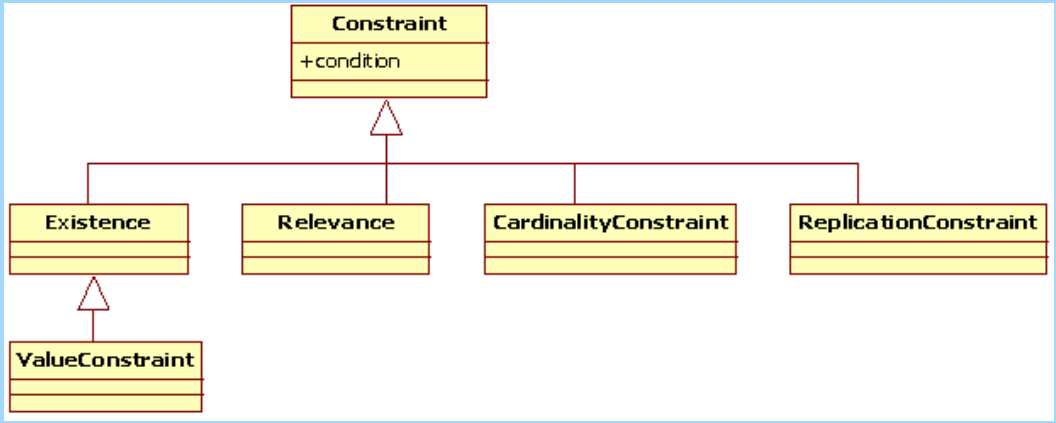
- A *DomainItem* is an atomic unit of data, i.e. which cannot be decomposed further.
  - The *itemType* denoting a data type, such as, but not limited to: Boolean, hour, date, natural, integer, real, character, string.
- A *DomainCollection* expresses the notion of data collection, i.e. how to collect individual domain items into a structured format. A concrete example of a collection of collections is a table. A concrete example of a collection of items is a comboBox.
- The *collectionType* denotes any data structure, such as, but not limited to: set, list, stack, tree, binary tree, shared tree, matrix.

Domain Model



- The *Constraint* relationship is self descriptive as it refers to constraints that affect to *DomainElements.* The *Constraint* is described with the *condition* attribute that specifies a condition attached to a constraint, i.e., a rule that must be fulfilled in the specification before or after the application of a *Constraint.*
- DomainElements A and B bounded by a functional dependency if and only if for each value of A, there is at most one value of B. A is called the determinant element, while B is called determined element. This relationship is described with the attribute semanticFunction that is the name of a method belonging to the semantic core of the interactive application, such as any method from a class.

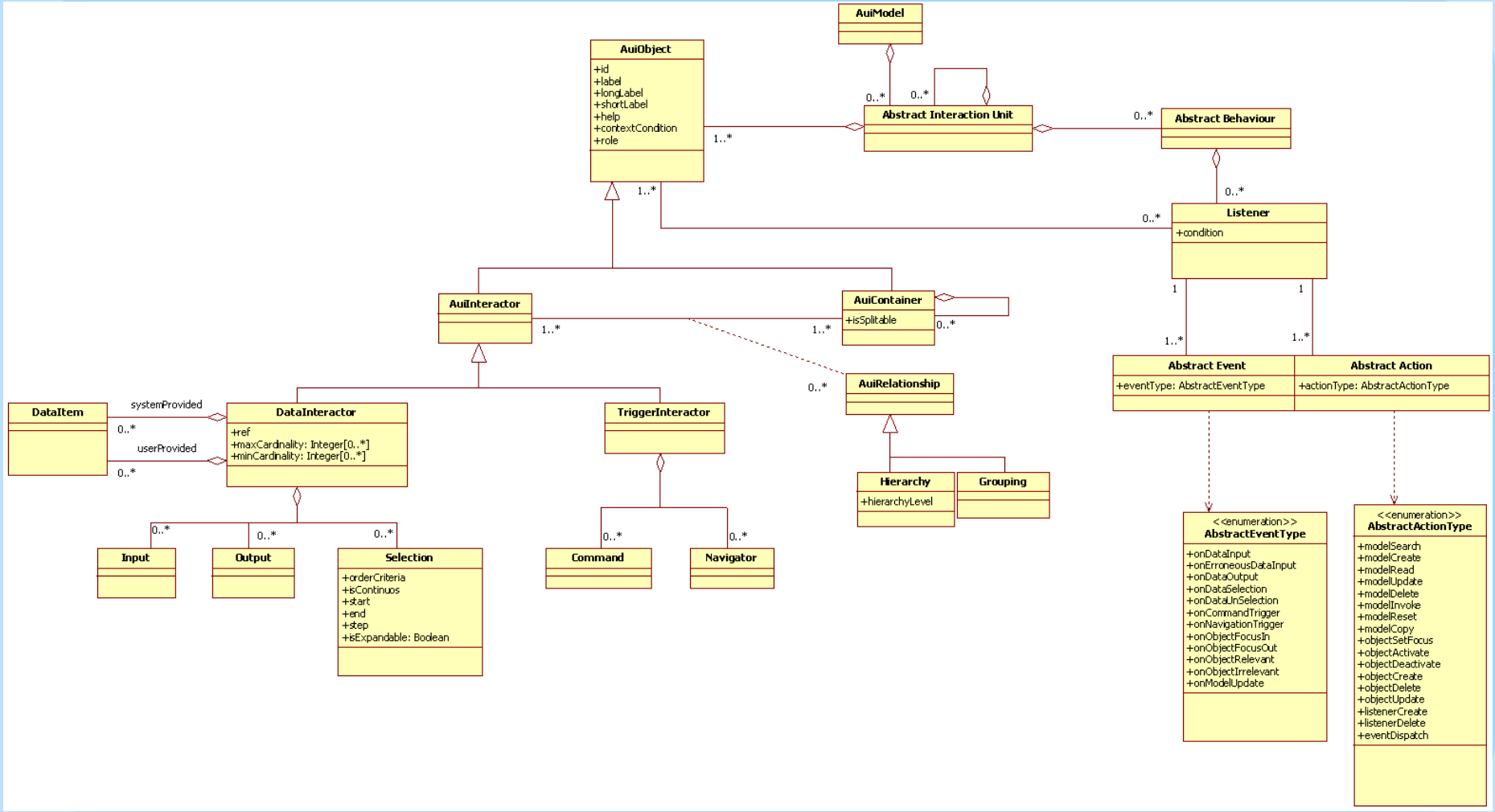
Domain Model



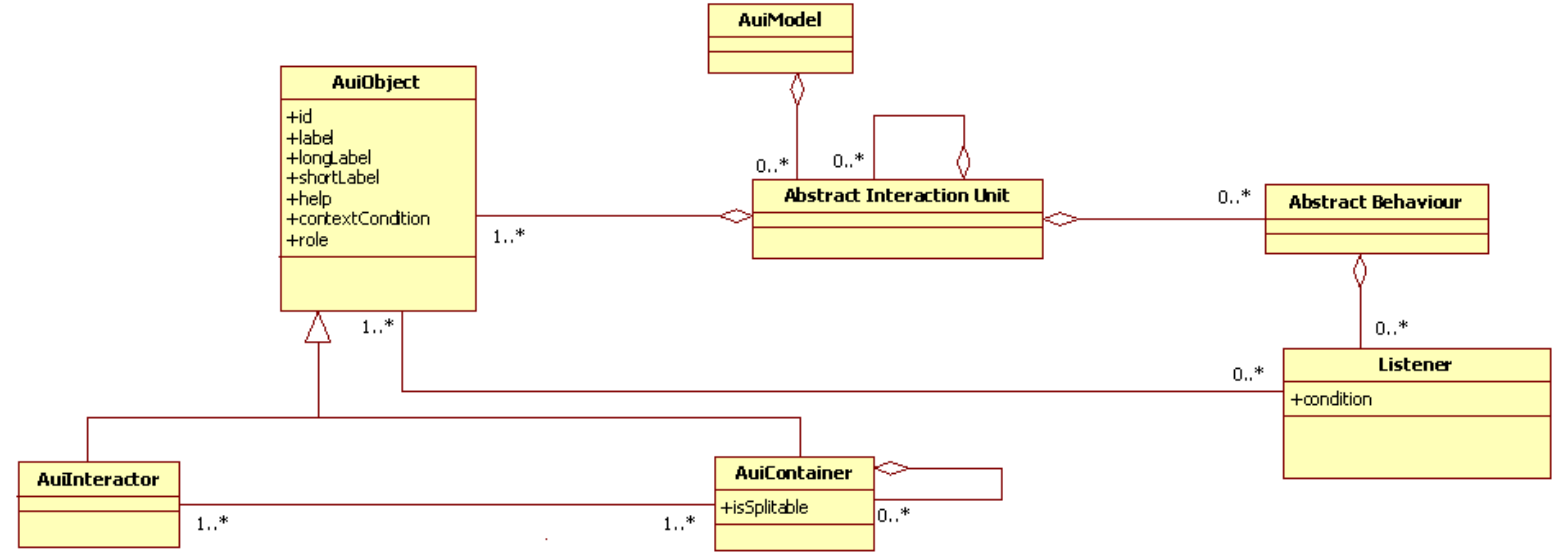
- *Existence* denoting the constraint for one *DomainElement* to consider the existence of another *DomainElement,* this is specialized in a *ValueConstraint,* for instance, to pay a flight ticket the user must have inserted their payment method.
- *Relevance* denoting the relevance or importance of a *DomainElement* compared to another *DomainElement.* For instance, in a webmail site the login box is more relevant than the welcome box.
- *CardinalityConstraint* denoting the cardinality constraint from one *DomainElement* to another. For instance, on a ecommerce application the shopping cart has a cardinality constraint to have at least one item when passing to the check out section.
- *ReplicationConstraint* denoting the constraint of replicating one *DomainElement* into another *DomainElement.*

Abstract Interface (device and modality independent

Abstract UI Model

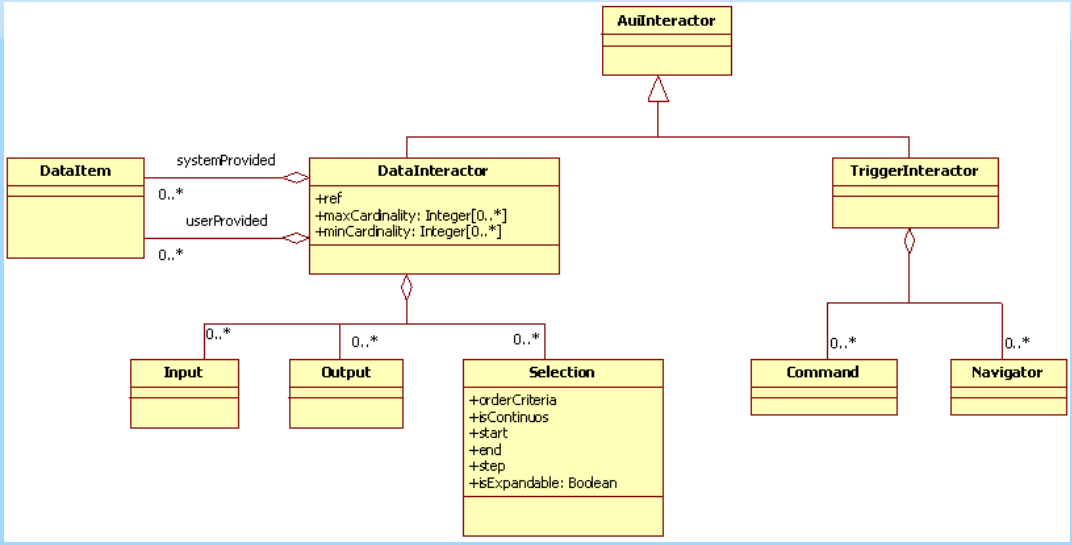


Abstract UI Model



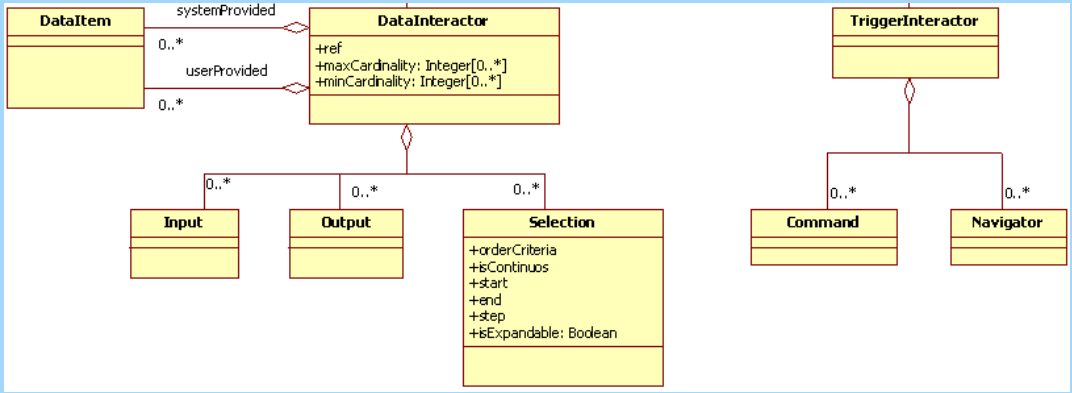
- The AUI model is composed of *AbstractInteractionUnits.*
- The *AbstractInteractionUnit* is composed of *AuiObject*s, *AbstractBehaviour,* and other *AbstractInteractionUnits.*
- The *AuiObject* (Abstract User Interface Object) is the root of the hierarchy of User Interface object. It is specialized in containers and interactors.
- The *AbstractBehaviour* corresponds to an abstract view of the behaviour of an *AbstractInteractionUnit*. The behaviour is an aggregation of *listeners,* which are associated to AuiObjects.

Abstract UI Model-AuiInteractor



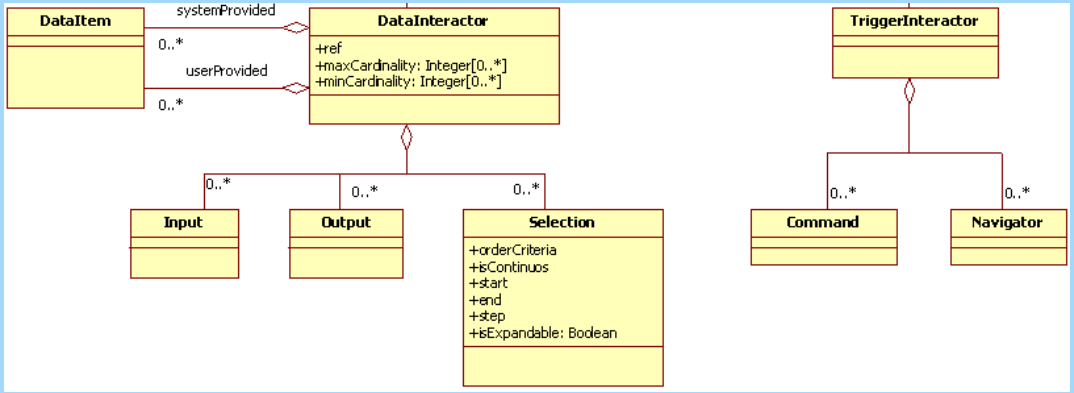
- *AuiInteractors* are described with the attributes: *ref* is a reference to the domain model element; *maxCardinality* represents the maximum cardinality for the *AuiInteractor*; and *minCardinality* represents the minimum cardinality for the *AuiInteractor*.
- The *DataInteractor* is an aggregation of the different types of elements that interacts with the user to present (*Output*) or obtain data (*Input* and *selection*).
- The *DateItem* is a concept used for the abstract behaviour model with two aggregation relationships (*systemProvided* and *userProvided*) that represents the system and user provided data directly linked to the *DataInteractor.*
- With this dentition there is a clear separation between the data it manipulates (*DateItem*) and the type of interactor (*Output*, *Input*, and *Selection*).

Abstract UI Model-AuiInteractor (cont...)



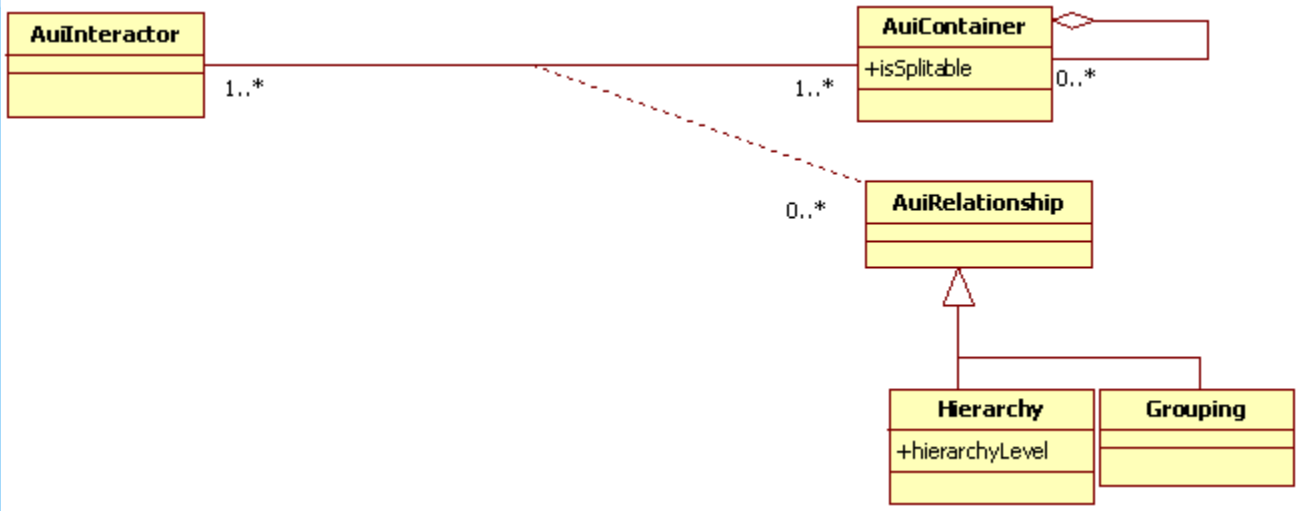
- The *Input* represents the input facet of a *DataInteractor*, which indicates that such interactor accepts the input of information from the user.
- The *Output* represents the facet of a *DataInteractor* aimed to present information to the user.
- The *Selection* is the representation of the facet of a *DataInteractor* that serves for the purpose of entering or displaying information.
- The *Selection* is described by the attributes: *orderCriteria* declares whether the values are ordered, *isContinuos* declares whether the values are continuous, *start* indicates the initial value for the set, *end* indicates the final value for the set; *step* indicates the difference between 2 contiguous values of the set; *isExpandable* indicates if it is possible to expand with new values

Abstract UI Model-AuiInteractor (cont…)



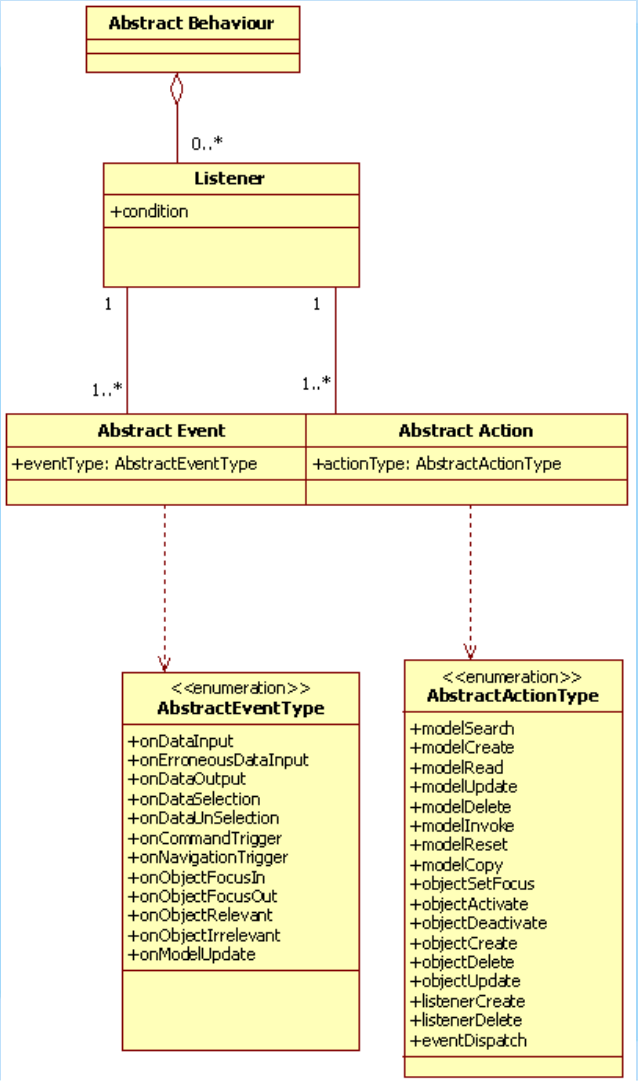
- The *TriggerInteractor* is an interactor that allows users to give orders to the system or to navigate to the functionalities they may need in a given moment. It can have two different facets: *Command* and *Navigator*.
- The *Command* represents a facet of a *TriggerInteractor* that serves for the purpose of command the system to perform a computation.
- The *Navigator* is a *TriggerInteractor* facet aimed to change the current presentation unit, thus allowing users to change to a different set of tasks to be accomplished.

Abstract UI Model-Relationships



- The *AuiRelationship* is an association class aimed to indicate explicitly the exact relationship between an *AuiContainer* and its contained *AuiInteractors*.
- In this sense an *AuiContainer* might have a *hierarchy* relationship indicating that the contained *AuiInteractors* form a *hierarchy,* which is described with a *hierarchy level*.
- The *Grouping* relationship declares a relationship between two or more *AuiInteractors* groped in the same *AuiContainer*.

Abstract UI Model- AbstractBehaviour

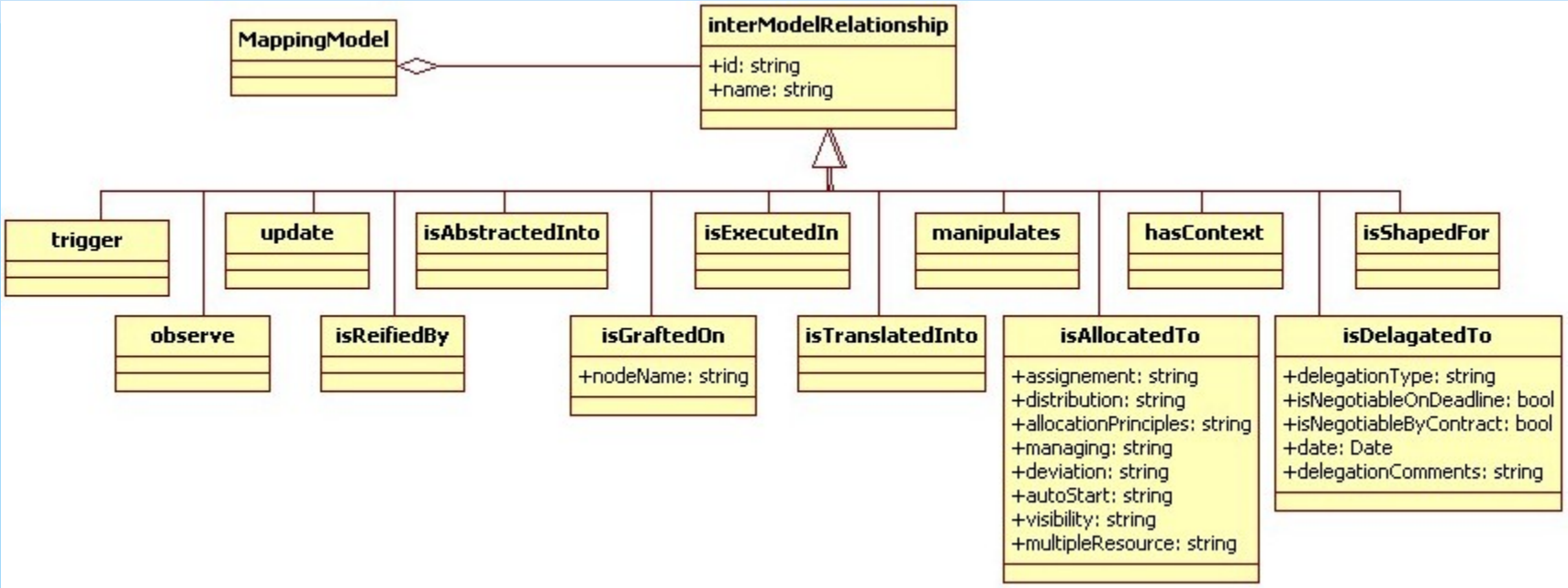


- The *AbstractBehaviour* corresponds to an abstract view of the behaviour of an *AbstractInteractionUnit*. The behaviour is an aggregation of *listeners.* A *listener* is associated to at least one *AuiObject.* The *listener* is seen as an ECA rule (on event if condition then action).
- A canonical set of *abstractevents* and *abstractactions* is proposed for the *events* and *actions.*
- Notice that both lists are not restricted to these values and can be expanded to any custom value.
- The *condition* attribute is just a textual expression.

Mapping Model

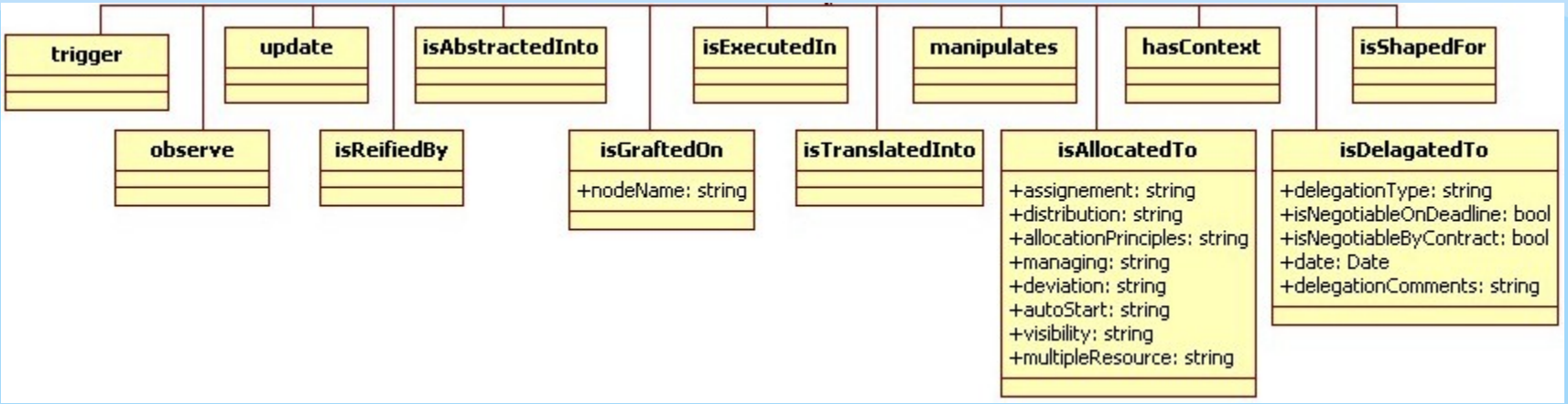
USIXML

Mapping Model



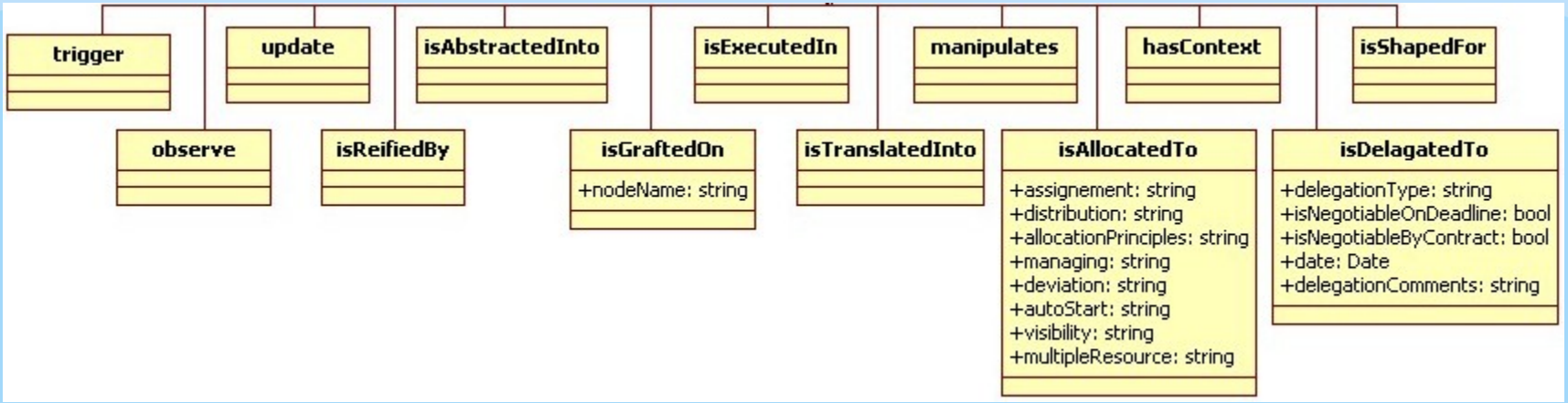
- The *MappingModel* contains a series of related mappings between models or elements of models. A *MappingModel* serves to gather a set of *interModelRelationship* that are semantically related.
- The *interModelRelationship* is any type of relationship established between one or many source models and one or many target models. A typical *interModelRelationship* 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.

Mapping Model



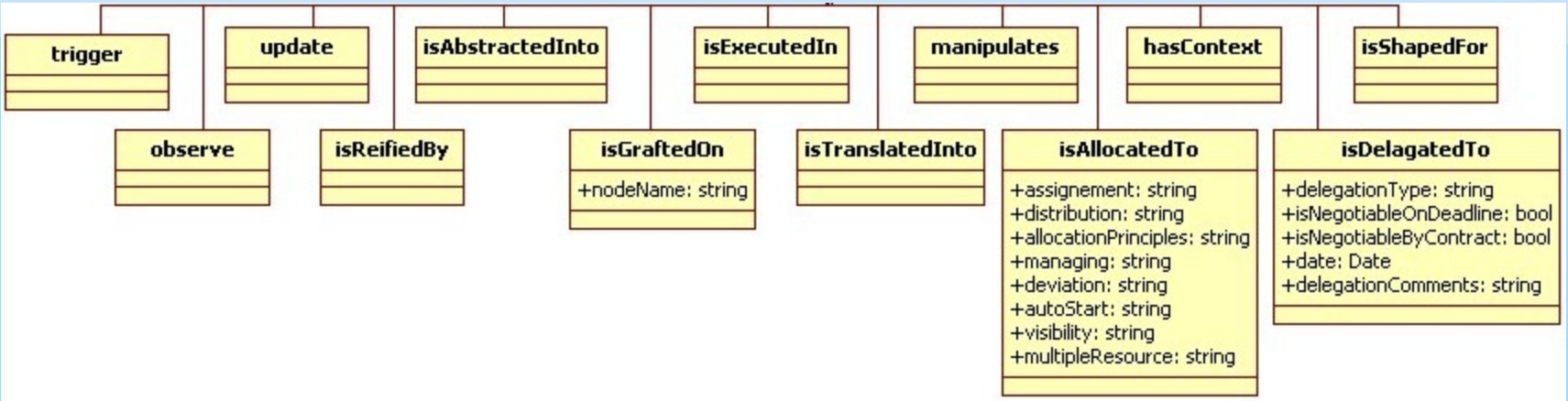
- A *Trigger* indicates a connection between a method of the domain model and a UI individual component (either at the abstract or at the concrete level).
- An *observe* is a mapping between any UI component (at abstract or concrete level) and a domain attribute or instantiated attribute (at run time). Observes enables to specify that a UI component observes a value from the related domain concept.
- An *update* 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.
- An *isReifiedBy* is a model relationship involving a source model that is reified into a target model. This relationship is the inverse of *isAbstractInto*.

Mapping Model



- An *isAbstractedInto* mapping maps a concrete user interface element onto an abstract element .
- An *isGraftenOn* mapping indicates that a task is grafted on another one. For instance, normal procedures are followed during the flight of a plane but the aircraft flight task is grafted into an aircraft flight during turbulence.
- An *isExecutedIn* mapping indicates that a task is performed through one or several abstract containers (ACs) and Abstract interactor.
- An *isTranslatedInto* mapping enables to provide a trace of the adaptation of one component in another.

Mapping Model



- A *manipulates* mapping maps a task onto a domain concepts i.e., a class, an attribute, an operation or any combination of these types. This relationship uses the task attribute 'centrality' which specifies the relative importance of the task to the execution of its corresponding task. This attribute 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).
- A *hasContext* mapping relates any UI model or, in some cases, model elements to the context(s) for which it is supposed to apply.
- A *isShapedfor* mapping allows to associate a plasticity domain to a CUI.
- A *isAllocatedTo* mapping indicates a task allocation to a resource.
- A *isDelegatedTo* mapping indicates a resource who is assigned to a task allocates it to another resource

Mapping Model-task Allocat

- The *isAllocatedTo* attribute indicates a task allocation to a resource. *Assignment* is the manner in which work items are advertised to specific resources for execution, including:

1. direct - Specify at design time the identity of the resource that will execute a task
2. deferred - Specify the identity of the resource that will execute a task until runtime
3. authorization-based - Specify the range of resources that are authorized to execute a task
4. separation of duties - Specify that two tasks within the same case, must be allocated to different resources
5. case handling - Allocate the work items within a given case to the same resource
6. retain familiar - Where several resources are available to undertake a work item, the ability to allocate a work item within a given case to the same resource that undertook a preceding work item
7. capability-based - Offer or allocate instances of a task to resources based on specific capabilities that they possess
8. history-based - Offer or allocate work items to resources on the basis of their previous execution history
9. hierarchy level-based - Offer or allocate instances of a task to resources based their position within the organization and their relationship with other resources

USIXML

Mapping Model-task Allocat

- *distribution* Is the manner in which newly created work items for a task are proactively offered or allocated to resources, including:

1. offer single-resource - Offer a work item to a selected individual resource
2. offer multiple-resources - Offer a work item to a group of selected resources
3. allocation single-resource - The ability to directly allocate a work item to a specific resource for execution
4. early distribution - The ability to advertise and potentially allocate work items to resources ahead of the moment at which the work item is actually enabled for execution
5. distribution on enablement - The ability to advertise and allocate work items at the moment they are enabled for execution
6. late distribution - The ability to advertise and allocate work items to resources after the work item has been enabled

Mapping Model-task Allocat

- The *allocationPrinciples* attribute is the manner in which work items are allocated to resources, including:

1. offer single-resource - Offer a work item to a selected individual resource
2. random allocation - Allocate work items to suitable resources on a random basis (e.g. give maintenance to sewing machine work item is allocated to a Technician on a random basis)
3. round robin allocation - Allocate a work item to available resources on a cyclic basis (e.g. Celebrate a mass work item is allocated to each available Priest on a cyclic basis)
4. shortest queue - Allocate a work item to the resource that has the least number of work items allocated to it (e.g. Make up model work item is allocated to the Makeup artist who has the least number of models allocated to him)

Mapping Model-task Allocat

- The *managing* attribute is the manner in which the work items are initiated by individual resources, including:

1. resource-initiated allocation - The ability for a resource to commit to undertake a work item without needing to commence working on it immediately
2. resource-initiated execution- allocated work item - The ability for a resource to commence work on a work item that is allocated to it
3. resource-initiated execution - offered work item - The ability for a resource to select a work item offered to it and commence work on it immediately
4. system-determined work queue content - The ability of the work flow engine to order the content and sequence in which work items are presented to a resource for execution
5. resource-determined work queue content - The ability for resources to specify the format and content of work items listed in the work queue for execution
6. selection autonomy - The ability for resources to select a work item for execution based on its characteristics and their own preferences

USIXML

Mapping Model-task Allocat

- The *deviation* attribute is when the normal sequence of state transitions for a workflow item is varied, including:

1. escalation - Offer or allocate a work item to a resource of group of resources other than those it has previously been offered or allocated to in an attempt to expedite the completion of the work item
2. deallocation - The ability to a resource to relinquish a work item which is allocated to it and make it available for allocation to another resource
3. stateful reallocation - The ability of a resource to allocate a work item to another resource without loss of state data

Mapping Model-task Allocat

- *deviation* (cont…)

١. stateless reallocation - The ability for a resource to reallocate a work item currently being executed to another resource without retention of state

٢. suspension/resumption - The ability for a resource to suspend and resume execution of a work item

٣. skip - The ability for a resource to skip a work item allocated to it and mark the work item as complete

٤. redo - The ability for a resource to redo a work item that has previously been completed in a case

٥. pre-do - The ability for a resource to execute a work item ahead of the time that it has been offered or allocated to resources working on a given case

USIXML

Mapping Model-task Allocat

- The *deviation* attribute denotes situations where execution of work items is triggered by specific events in the lifecycle of the work item or the related process definitions, including:

1. commencement on creation - The ability for a resource to commence execution on a work item as soon as it is created (e.g. Update Database work item commences execution as soon as it is created)
2. commencement on allocation - The ability to commence execution on a work item as soon as it is allocated to a resource (e.g. begin the Type letter work item as soon as it is allocated to a Secretary)
3. piled execution - The ability of the system to initiate the next instance of a task list once the previous one has completed (e.g. the next Calculate the taxes work item can commence immediately after the previous one has finished)
4. chained execution - The ability of the workflow engine to automatically start the next work item in case once the previous one has completed (e.g. immediately commence the next work item in the Emergency rescue coordination process when the preceding one has completed)

USIXML

Mapping Model-task Allocat

- The *visibility* attribute ere the scopes in which work item availability and commitment are able to be viewed, including:

1. configurable unallocated work item visibility - The ability to configure the visibility of unallocated tasks by users (e.g. the Lawyer can only see the unallocated work items that may be subsequently allocated or offered to him)
2. configurable allocated work item visibility - The ability to configure the visibility of allocated task by users (e.g. the Doctor can view the allocated work items list for the day)

Mapping Model-task Allocat

- The *multipleResources* attribute represents situations where there is often many-to-many correspondence between the resources and work items in a given allocation or execution., including:

1. simultaneous execution - The ability for a resource to execute more than one work item simultaneously (e.g. the Bank teller can conduct multiple Foreign exchange work items at the same time)
2. additional resources - The ability for a given resource to request additional resources to assist in the execution of a work item that they are currently undertaking (e.g. the Builder has requested additional Builder's labourer before continuing with the Built wall work item)

Mapping Model-task Allocat

- The *multipleResources* attribute represents situations where there is often many-to-many correspondence between the resources and work items in a given allocation or execution., including:

1. simultaneous execution - The ability for a resource to execute more than one work item simultaneously (e.g. the Bank teller can conduct multiple Foreign exchange work items at the same time)
2. additional resources - The ability for a given resource to request additional resources to assist in the execution of a work item that they are currently undertaking (e.g. the Builder has requested additional Builder's labourer before continuing with the Built wall work item)

Mapping Model-task Delegation

- A *isDelegatedTo* mapping is 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.

Attributes:

delegationType: Describe the type that a resource makes a delegation of his task. It could be by negotiation, by assignment or by tender.

isNegotiableOnDadline : Indicates if the task is negotiable on the time limit for its execution

isNegotiableOnContract : Indicates if in the negotiation there is a contract in which exists some conditions for execute a task.

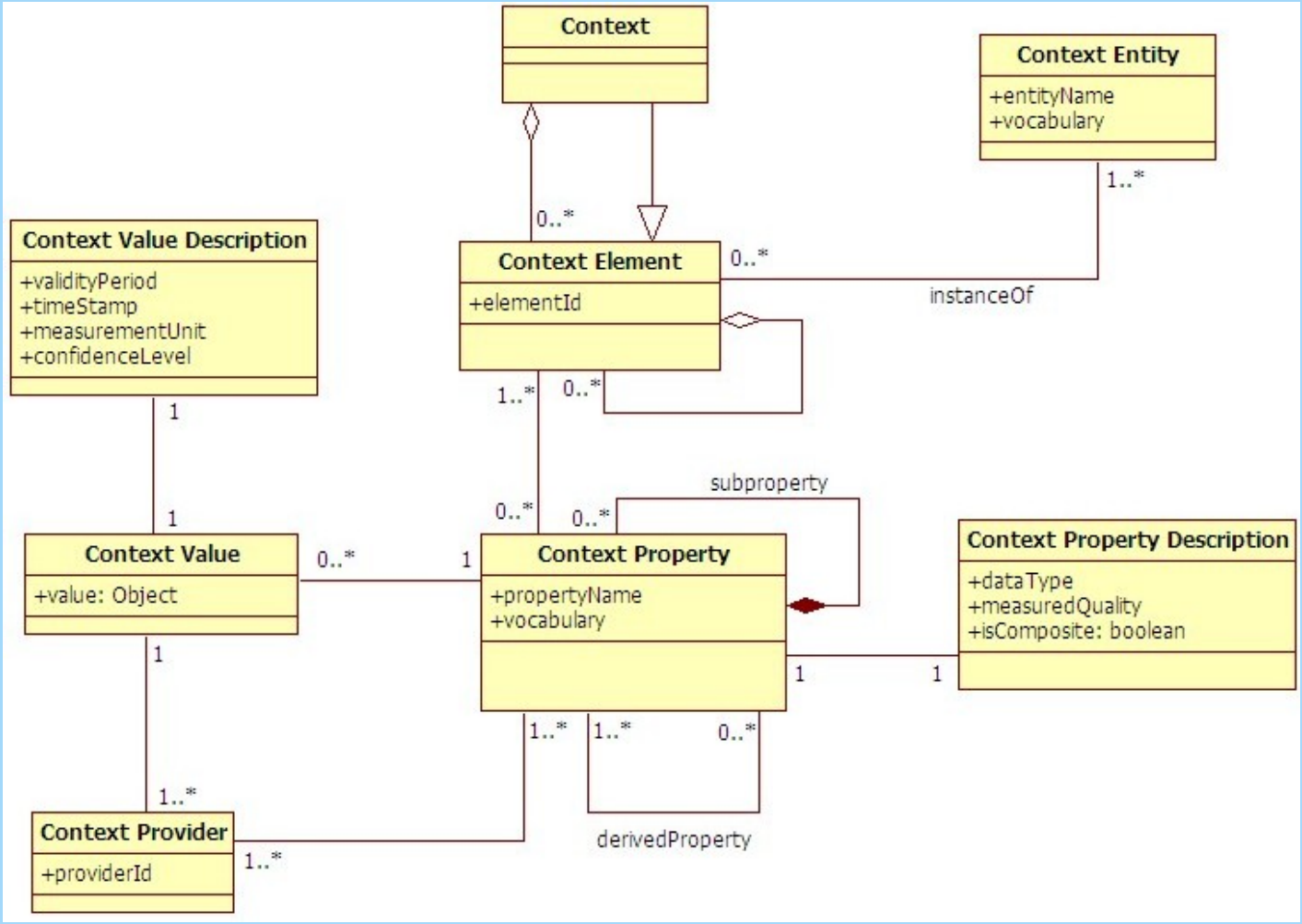
date: Is the date in which the delegation was done

delegationComments : In delegationComments is possible to add extra information, for instance, some observation about the task to be delegated.

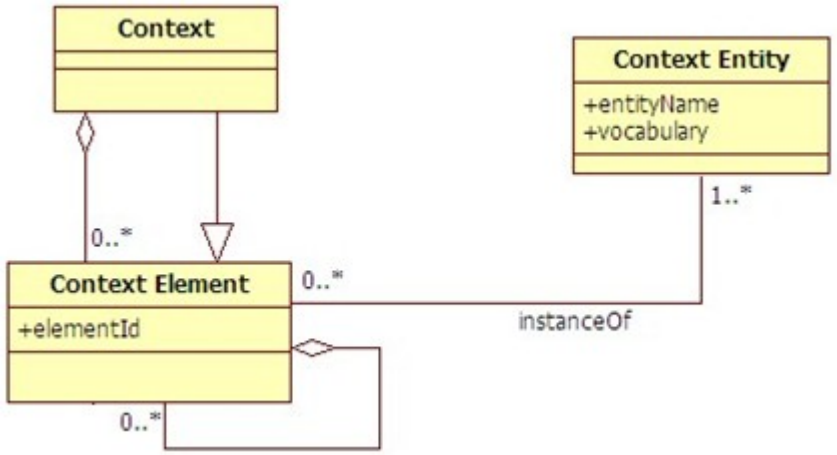
Concrete Interface (device and/or modality dependent. Still ongoing work....

Context Model

Context Model

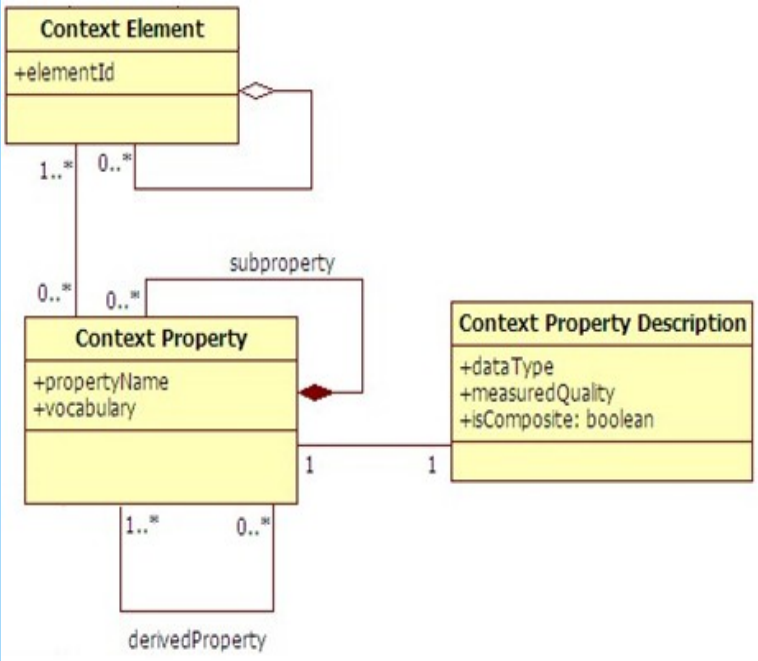


Context Model



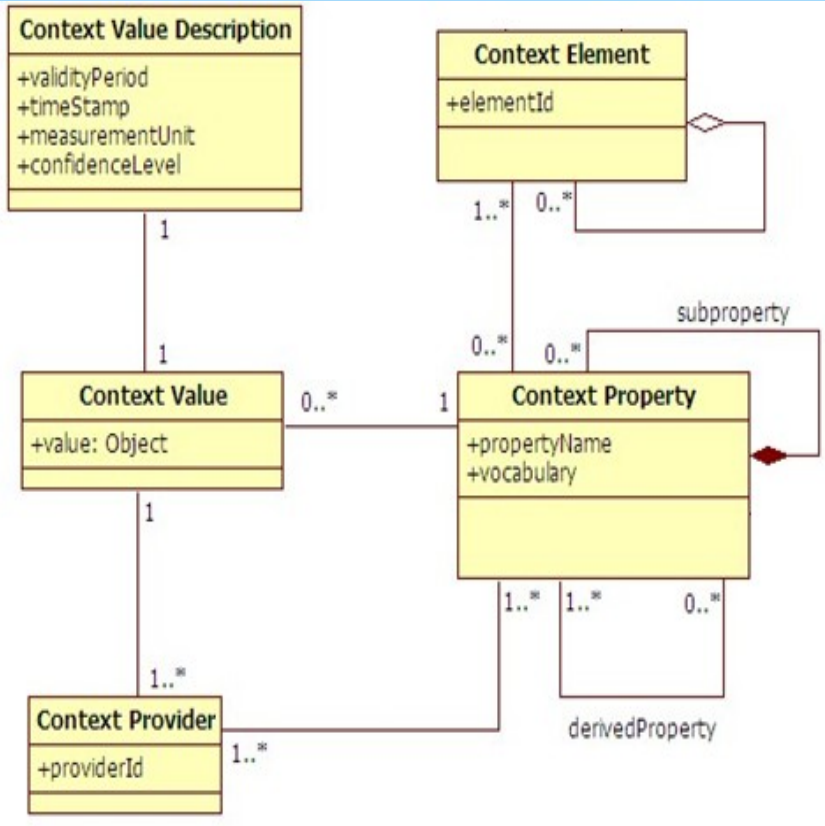
- The *Context* represents the Context itself as a whole composed by the aggregation of different Context Elements (modeled as instances of Context Entities).
- The *Context Entity* represents types of context entities. It can be 'User', 'Object', 'Device' and the like.
- The *Context Element* denotes an specific instance of a Context Entity. Context Elements have different properties, which values are actually the information that characterize their situation.

Context Model



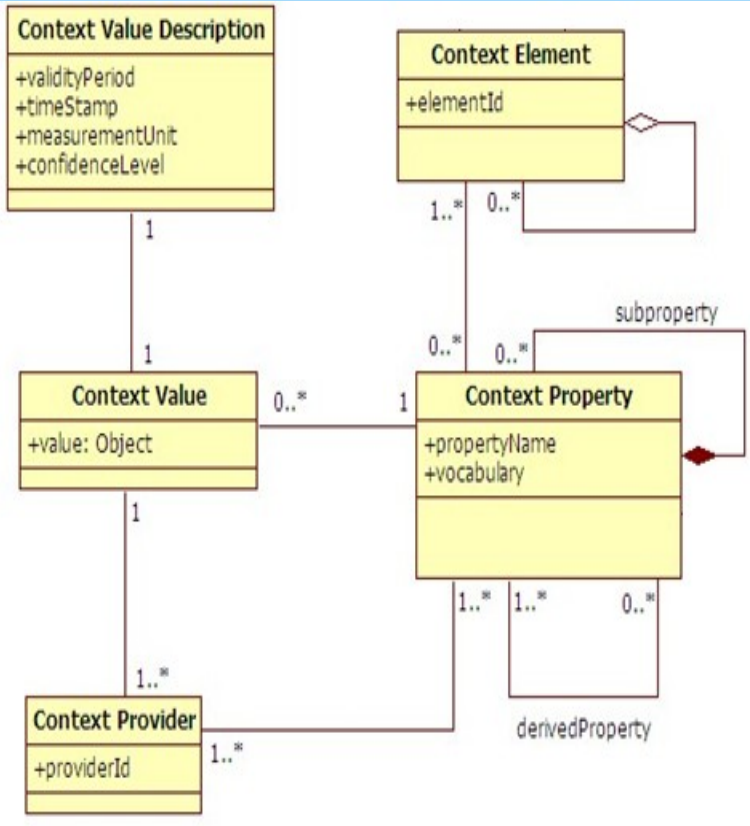
- The *Context property* represents a property of a Context Element, which represents a characteristic of such Context Element or information about its situation. A Context Property may have zero or more values. Examples of properties are: 'position' or 'cpuSpeed'. There can be properties composed by other subproperties. For example, the 'position' property may have three subproperties: 'latitude', 'longitude' and 'altitude'.
- The *Context Property Description* represents the description of a Context Property and incorporates different attributes for conveying such metadata. It is described by the *datatype* of the property and the *measuredQuality*, representing the quality that a property may represent. For example, 'length' for the 'width' property.

Context Model



- The *Context Value* represents a value for a Context Property. A *Context Value* might be associated to one or more Context Providers, the value is conveyed by the attribute *value*.
- The *Context Provider* represents a source of context information i.e. it is capable of providing the values of Context Properties.

Context Model



- The *Context Value Description* represents the description of the value of a Context Property and incorporates different attributes for conveying such metadata. Attributes:
  - *validityPeriod*: Indicates the period of time (in seconds) for which this property value will be valid. After that period (counted from the timestamp) the value will be considered as out of date.
  - *timeStamp*: a timestamp that indicates the precise moment in time when the property value was obtained
  - *measurementUnit*: The unit that might correspond to the value. For example, "kilo" for a "weight" property value.
  - *confidenceLevel*: This property indicates the level of confidence for the property value.