Multi-Dimensional Context-Aware

Adaptation of Service Front-Ends

Project no. FP7 – ICT – 258030

Deliverable D.3.2.2
ASFE-DL: Semantics, Syntaxes and Stylistics (R2)

Due date of deliverable: 30/09/2012

Actual submission to EC date: 30/09/2012

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination level		
[PU]	**[Pubic]**	**Yes**

Document Information	
Lead Contractor	TID
Editor	CNR-ISTI
Revision	v.1.0 (26/09/2012)
Reviewer	SAP
Approved by	TID
Project Officer	Michel Lacroix

Contributors	
Partner	**Contributors**
ISTI	**Fabio Paternò, Carmen Santoro, Lucio Davide Spano,**
UCL	**Vivian Motti**

Changes			
Version	**Date**	**Author**	**Comments**
1	12/09/2012	ISTI	First document draft
2	20/09/2012	ISTI	First consolidated version
3	26/09/2012	SAP	First revision
4	27/09/2012	ISTI	Final version

Executive Summary

This report describes the intermediate state of the work in SERENOA for creating a model-based Description Language (DL) for User Interfaces (UI). The deliverable discusses an updated version of the Advanced Service Front End Description Language (ASFE-DL) for the Abstract level, defined in the previous release of this deliverable. The new version of ASFE-DL for the Abstract UI level takes into account the points that have been raised not only inside the SERENOA consortium, but also the ideas that have been discussed in the W3C Model-Based User Interface (MBUI) group. In addition, this document provides a first version of ASFE-DL that covers the Concrete Level. The document ends with a set of conclusion and with a set of actions for further improving the languages state.

Table of Contents

1 Introduction

1.1 Objectives

In this document, we will report on the updates to the first version of the Description Language (DL) for User Interfaces (UI) that it is used in Serenoa. The language is compliant with the Cameleon Reference Framework [Calvary2002]. Therefore, the modelling of a UI is defined through a set of levels of abstraction (Task & Concepts, Abstract User Interface, Concrete User Interface, Final User Interface). The previous version of the ASFE-DL only covered the first version of the Abstract User Interface (AUI) level, which provides a platform and technology independent description of a UI. In this deliverable, we update the meta-model for this level, taking into account not only the Serenoa requirements, but also taking as input the discussion in the W3C Model Based User Interface (MBUI) group. In addition, the document defines also a first version of the ASFE-DL for the Concrete Desktop level. Finally, we describe the plan for future improvements of the languages.

1.2 Audience

Being a public deliverable, this document will be available outside the confines of the project's consortium and is intended to be of interest to the following parties:

a) Members of the consortium, who will find here a detailed description of the fundamentals of the Description Language that the project is to use in the future.
b) Researchers in the relevant fields: adaptation of SFEs, UI theorists, descriptive languages and medium-scale project software engineering.
c) EC officials that will use the information in this document as an account of the activities taken in the project tasks that inform this work.

1.3 Related documents

- **D1.2.1 Architectural Specifications (R1)**, which specifies the Reference Architecture context-sensitive SFEs based on the Serenoa technology, and how the different engineering pieces fits together.
- **D2.1.2 CARF and CADS (R2)**, which includes a second release of the Reference Framework for Context Aware Adaptation and Service Front Ends. The latter are described in Serenoa through the ASFE-DL language
- **D3.1.2 Reference Models Specification (R2),** which includes the second release of a set of Reference Models for the adoption process of Serenoa.
- **D3.2.1 ASFE-DL: Semantics, Syntaxes and Stylistics (R1)** which describes the first version of the ASFE-DL. **D3.3.1 AAL-DL: Semantics, Syntaxes and Stylistics (R1),** which specifies the first version of the language for expressing advanced adaptation logic (AAL). The actions described for adapting the User Interface may change properties of the UI at different levels of abstraction.
- **D4.5.2 Authoring Environment (R2)**, which provide the second update on the development status of the authoring environment and analysis tools for the creating adaptive SFEs. This of course includes also creating ASFE-DL models.
- **D5.1.1 Serenoa Framework (R1**): which describes the first version of the Serenoa Framework, including the design of the common interfaces, the integration of the software components and the interface definitions and code documentation.
- **D6.2.1) Standardization Actions Report (Update 1)**, which describes the standardization effort on the W3C MBUI working group. The first version of the ASFE-DL language has been submitted for the standardization.

1.4 Organization of this document

Section 1 describes the scope and the organization of this document. Section 2 summarizes the inputs provided by the submission W3C working group and provides a comparison of the concept defined in the ASFE-DL and those defined in the others submissions, in order to find a common set. Section 3 defines the changes to the first version of the ASFE-DL abstract UI language, while Section 4 introduces the first version of the Concrete Desktop meta-model. Section 5 presents the conclusions and future work.

2 MBUI WG Submissions Analysis

In this section we briefly summarize the different languages that have been submitted for standardization to the W3C Model-Based User Interfaces Working Group (MBUI-WG), see http://www.w3.org/wiki/Model-Based_User_Interfaces). The participation at the working group with the submission of the ASFE-DL, together with the discussion of the various concepts included in the other languages submitted had a twofold impact on the design of the next version of the ASFE-DL. On the one hand, comparing ASFE-DL with the other languages gave us the possibility to compare the modelling concept coverage of the Serenoa language with respect to the other state of the art submissions. On the other hand, the discussion of abstract techniques for modelling UIs with experts that are external to the project, provided a useful input for the new version of the language.

After the description of the key concepts of each submission, we provide the description of a common set of modelling concepts, highlighting which ones have been identified in the various languages that cover the Abstract User Interface (AUI) level of the Cameleon Reference Framework [Calvary2002]. Such analysis has been used as input for the updates to AUI of the ASFE-DL, which are described in Section 3.

The languages described in the following sections represent the state of the art of the research in the Model-Based approaches for User Interfaces.

2.1 Submissions Summary

2.1.1 ASFE-DL

This is the meta-model of the language described in the deliverable [D3.2.1], as it was submitted to the W3C MBUI WG. Here we summarize its main modelling concepts:

- **AbstractUIModel**, represents the model of a given UI, with the specification of its structure and behaviour at the abstract level.
- **DataModel**, defines the data types manipulated by the user interface, which allow maintaining the state of the interaction with the user.
- **AbstractInteractionUnit** (AIU), represents a part of the application UI that should be presented to the user at once.
- **Connection,** expresses the possibility to navigate from one AIU to another
- **AbstractInteractor**, the base class for all the different interaction objects. Different types of interactors are defined in ASFE-DL:
  - **Selection**, which allows the user to select one (*SingleChoice*) or more options (*MultipleChoice*) from a predefined set of choices.
  - **Edit**, which allows the user to manually modify an input value
  - **Only output,** which present information to the user
  - **Control**, which allows the user to trigger actions.
- **AbstractRelationship,** the base class of all the relationships among abstract interactors. Different types of relationships are defined in ASFE-DL:
  - **Grouping**, which represents a generic group of interactors
  - **Ordering,** which represents an ordering relationship among a group of interactors
  - **Hierarchy**, which represents a hierarchical relationship among a group of interactors
  - **Repetition,** which represents the template for a list of interactors that have to be repeated for each element of a dynamic list (e.g. a list of search results)
  - **Dependency**, which defines a dependency relation between 1 interactor/interactor group and N interactors/interactor groups.
- **AbstractDialogModel**, defines the behaviour of the UI, composing events through temporal operators.

Figure 1 shows the UML class diagram for the meta-model.

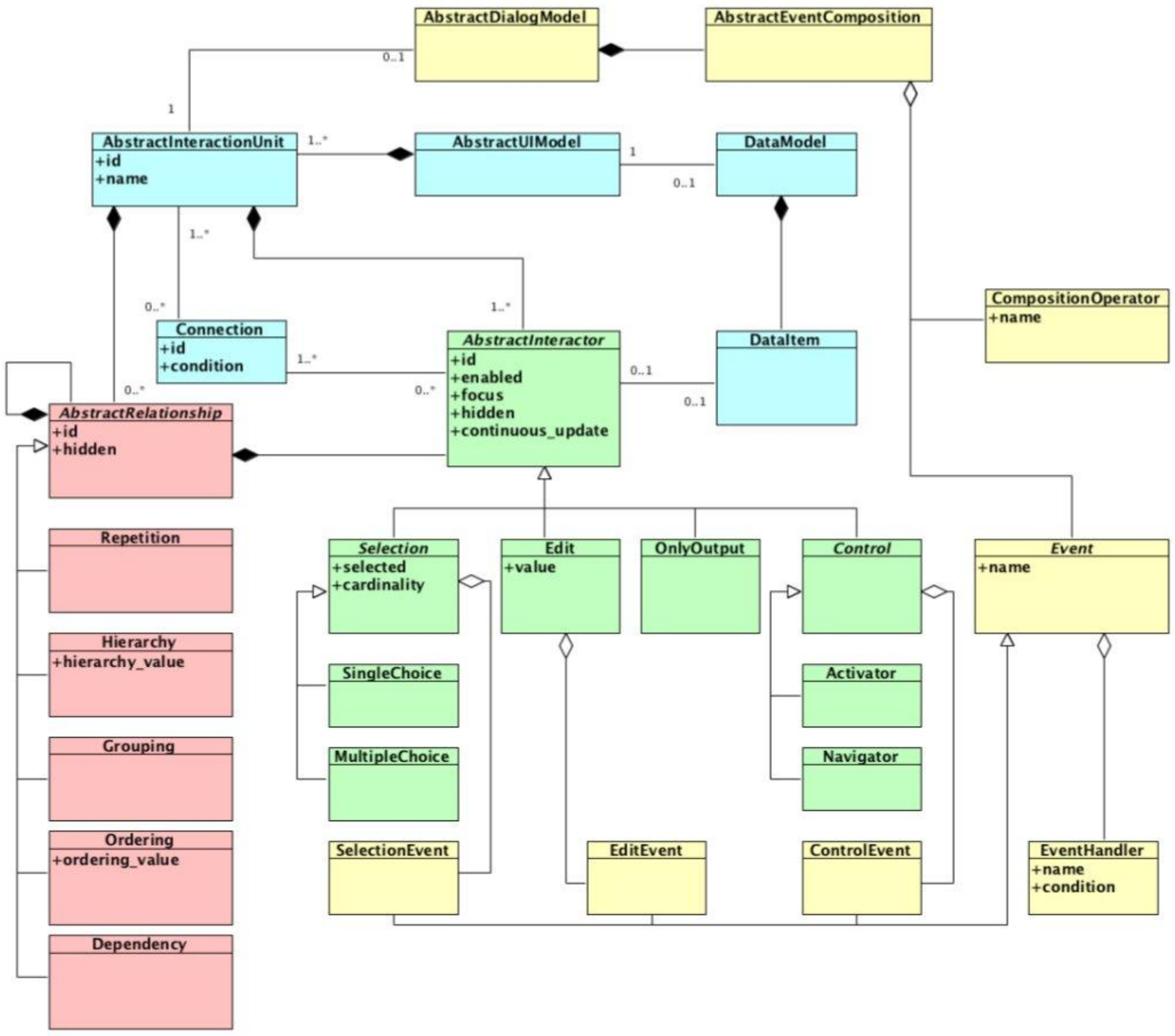

Figure 1 UML diagram of the ASFE-DL ver. 1, Abstract Level

2.1.2 MARIA

MARIA (Model-based language for Interactive Applications) [Paternò2009], is a universal, declarative, multiple abstraction-level, XML-based language for modelling interactive applications in ubiquitous environments, which is compliant to the Cameleon Reference Framework. It provides model for the Abstract and the Concrete levels (Desktop, Mobile, Multimodal Desktop, Multimodal Mobile, Vocal) .

Figure 2 shows the meta-model for the Abstract User Interface. The following is a brief list of the modelled concepts:

- **Interface**. Represents the model of a given application
- **Presentation**. Represents a set of interactors and interactor compostions that are logically connected representing a single unit to be presented to the user.
- **Data Model**. Defines the data types manipulated by the user interface.
- **Connection**. Defines a link between two presentations. The user can move from the source presentation to the target one.
- **Grouping.** A generic group of interactor elements.
- **Relation.** A group where two or more elements are related to each other.
- **Composite Description.** It represents a group aimed to present contents through a mixture of

Description and Navigator elements.

- **Repeater.** It is used to repeat the content according to data retrieved from a generic data source.
- **Interactor** An interactor represents every type of interaction object. The following is the list of possible interactor types:
    - **Selection**: Allows the user to select one or more values among the elements of a predefined list.
    - **Edit**: Allows the user to manually edit the object represented by the interactor (text, number, position etc.)
    - **Control**: Allows the user to switch between presentations or to activate UI functionalities
    - **Only output**: Represents information that is presented to the user and it is not affected by the user's actions.
- **Dialog Model.** The dialog model contains constructs for specifying the dynamic behaviour of a presentation, specifying what events can be triggered at a given time. The dialog expressions consists of expressions of UI events connected using CTT operators in order to define their temporal relationships.
- **Event** Each interactor has a number of associated events that allow the specification of UI reactions triggered by the user interaction.

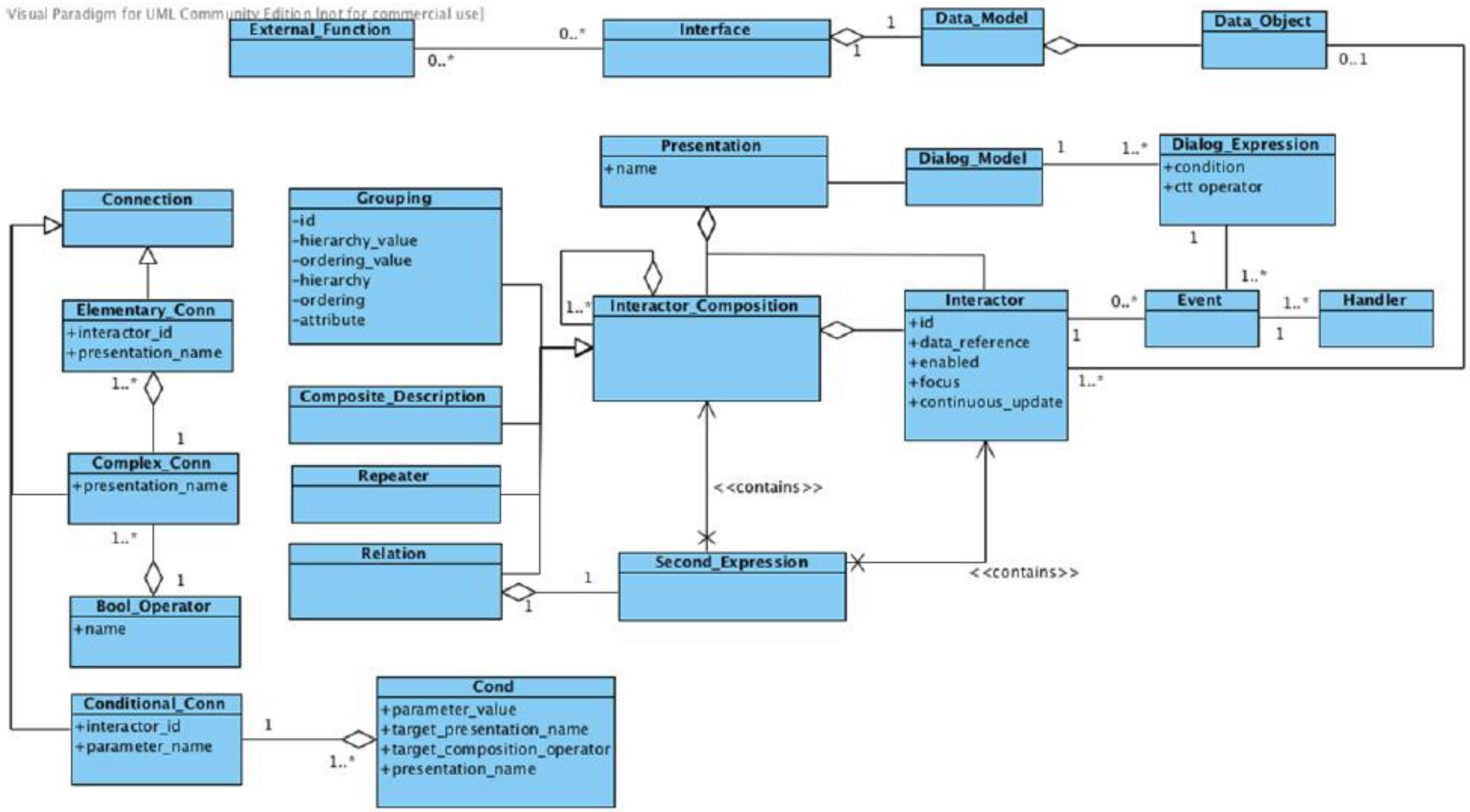

Figure 2 MARIA

2.1.3 MINT Abstract Interactor Model

Figure 3 shows the meta-model of the Abstract Interactor Model [Feuerstack2011], implemented in the MINT platform. The language is particularly targeted in order to support a multimodal setup, which combines at least one media with several modes. Thus, parts of the interface interactors are output-related, whereas others serve to address the modes used by the user to control the interface . At the abstract level, it has the peculiar feature to strictly separate the interactors used for the input from those used for the output, in order ease the different allocations of the modalities at the concrete level.

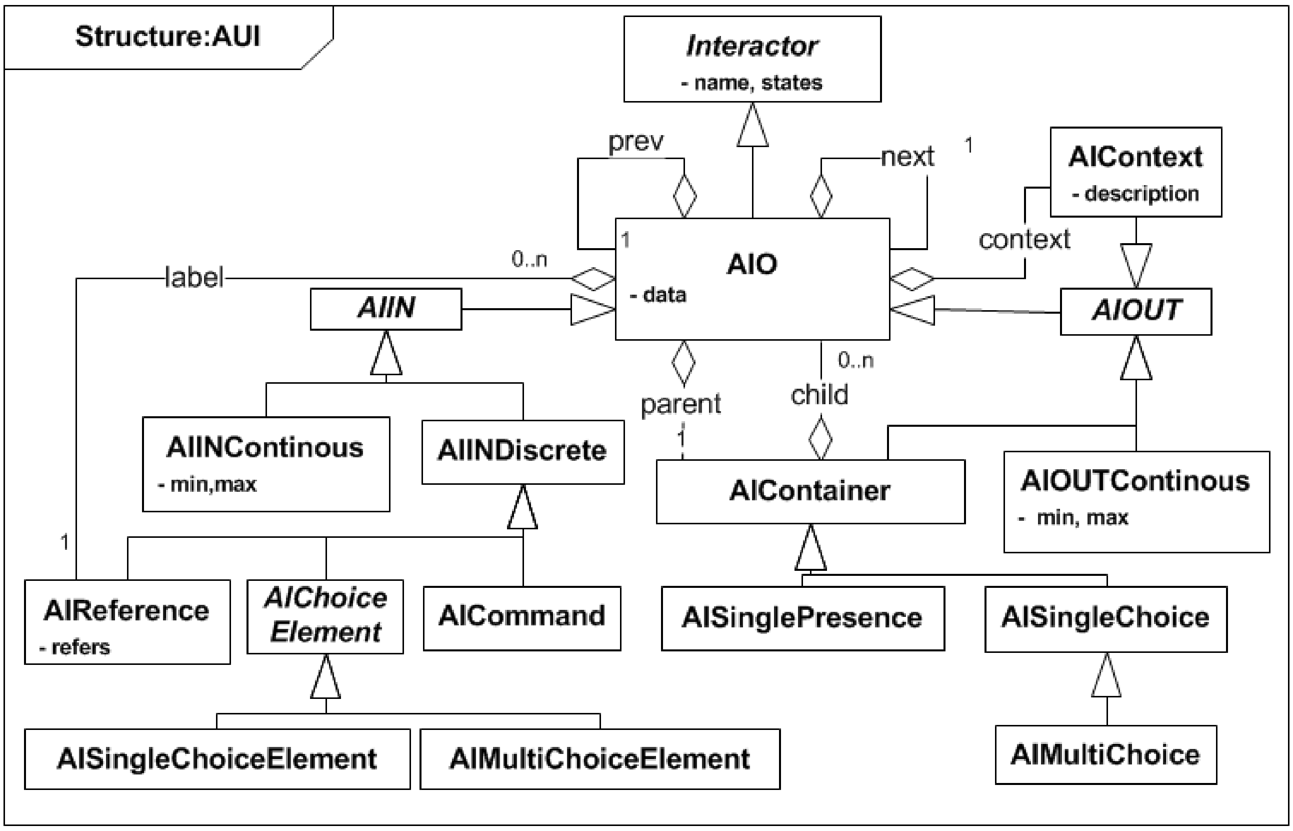

Figure 3 MINT Abstract Interactor Model

The following is the list of the main concepts in the meta-model.

- **AIIN**: interactors that are related to the input
    - **AIIN continuous**: input interactors with a continuous nature. Continuous outputs are a graphical progress bar, a chart, a tone that changes its pitch, or a light, which can be dimmed, for instance
    - **AIIN discrete**: input interactors with a discrete nature. Discrete user inputs could be performed by commands that are issued by voice, or by pressing a physical or virtual button, for instance. It is possible to consider choosing one or several things from a list as user input as well (AISingleChoice and AIMultiChoice respectively).
- **AIOUT**: interactors that are related to the output. This category is strictly separated fromt the user input, and there are some implications of this strict separation: For instance user choices are required to be separated to two different elements: a list of elements to choose from is considered as output (since it just signalizes the information that a user can choose together with a grouping of interactors), the current interactors to choose from are modeled separately as AIIN.
- **AIContainer:** handles the grouping of. AIContainers can be specialized to realize single (AISingleChoice) or multiple choices (AIMultiChoice) and are derived from the discrete output interactor.
- **AIContext.** add contextual information that helps the user to control the interface or to understand the interface interactor. This could be, for instance, a tooltip, a picture or a sound file.
- **Behavior.** AIM distinguishes three different types of behavior: (1) The behavior of an interactor, (2) the navigation between interactors, (3) The behavior of interactors in relation to other interactors in (a) the same model and (b) to an interaction resource.

- **Interactor behavior specification.** The interactor behaviour is specified using statecharts that are specified in SCXML and can be directly executed and tested (we use RSpec).
- **Interactor navigation / Dialogue navigation.** Abstract Interactors contain a rudimentary basic navigation that defines the minimal navigation capability and needs to be setup mode and media independent. This is done inside the statecharts and is limited to previous/next/parent/child navigation for the abstract level. Concrete interactor models could further refine the navigation to support e.g. up/down or left/right navigation schemes. Further on, there is the option for direct navigation (direct entry) that is modelled by the AIReference interactor and enables direct navigation to a specific interactor.
- **Interactor behavior in relation to others** We design this behavior in a separate model, that we call the Multimodal Interactor Mapping Model (MIM). The model distinguishes between Observations (events), Operators, and Actions.

2.1.4 UseDM

UseDM [Seissler2011] describes a meta-model and XML format for defining abstract dialog models for context-sensitive interactive systems. Figure 4 shows the entities of the meta-model.

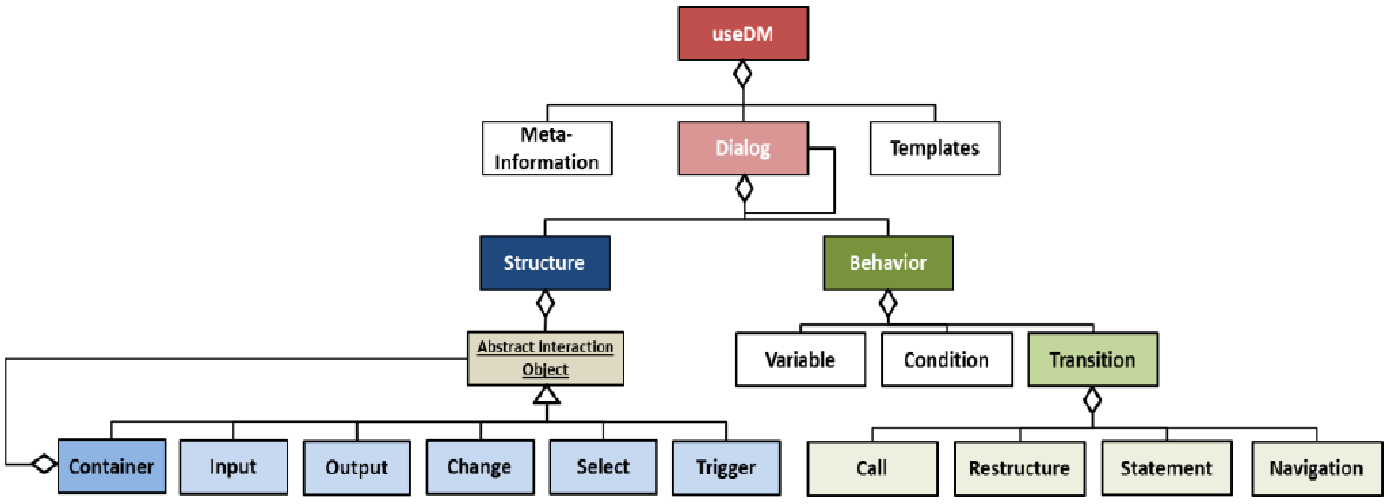

Figure 4 UseDM meta-model

- **UseDM model.** Consists of meta information, templates and a dialog which is refined by further dialogs. A dialog is split into structure and behaviour
- **Structure.** Specifies the abstract interaction objects as well as their hierarchical structure. UseDM specifies six heterogeneous abstract interaction objects that support the specification of a modality-independent information exchange between the user and the user interface.
- **Dialog.** The dialog model represents the abstract user interface representation and groups all interactions that can be performed at one point in time in several dialogs. Furthermore, the dialog model is used to specify the interaction and navigation behaviour of the user interface
- **Container** A container is used to hierarchically structure the abstract interaction objects.
- **Input.** The input element is used to specify the data input from the user perspective. The data can be of any type, e.g. text or numbers.
- **Output.** The output element is used to specify the data output from the user perspective. The data can be of any type, e.g. text or numbers.
- **Change.** The change element is used to specify the increment/decrement of a numerical value within a predefined interval
- **Select.** The select element is used to specify the inclusive/exclusive selection of a value from a set of values.
- **Trigger.**The trigger element is used to specify a command from the user perspective. This might be a function call or a navigation trigger.

- **Behavior.** Used to specify the user interface behavior. Core of the behavior description is an event-based transition model that supports the call of backend functions, the execution of restructure and statement expressions that have an effect on the interaction objects in the dialog's structure section and absolute/relative navigations. While variables are used to store global and local data within the user interface, conditions serve as 'guards' to specify under which conditions a transition might be executed.

2.1.5 UsiXML

User Interface eXtensible Markup Language (UsiXML) [UsiXML11] is a formal Domain-Specific Language (DSL) used in Human-Computer Interaction (HCI) and Software Engineering (SE) in order to describe any user interface of any interactive application independently of any implementation technology. Figure 5 shows the UML class diagram of the Abstract User Interface meta-model and it is followed by the list of the main modelling concepts.

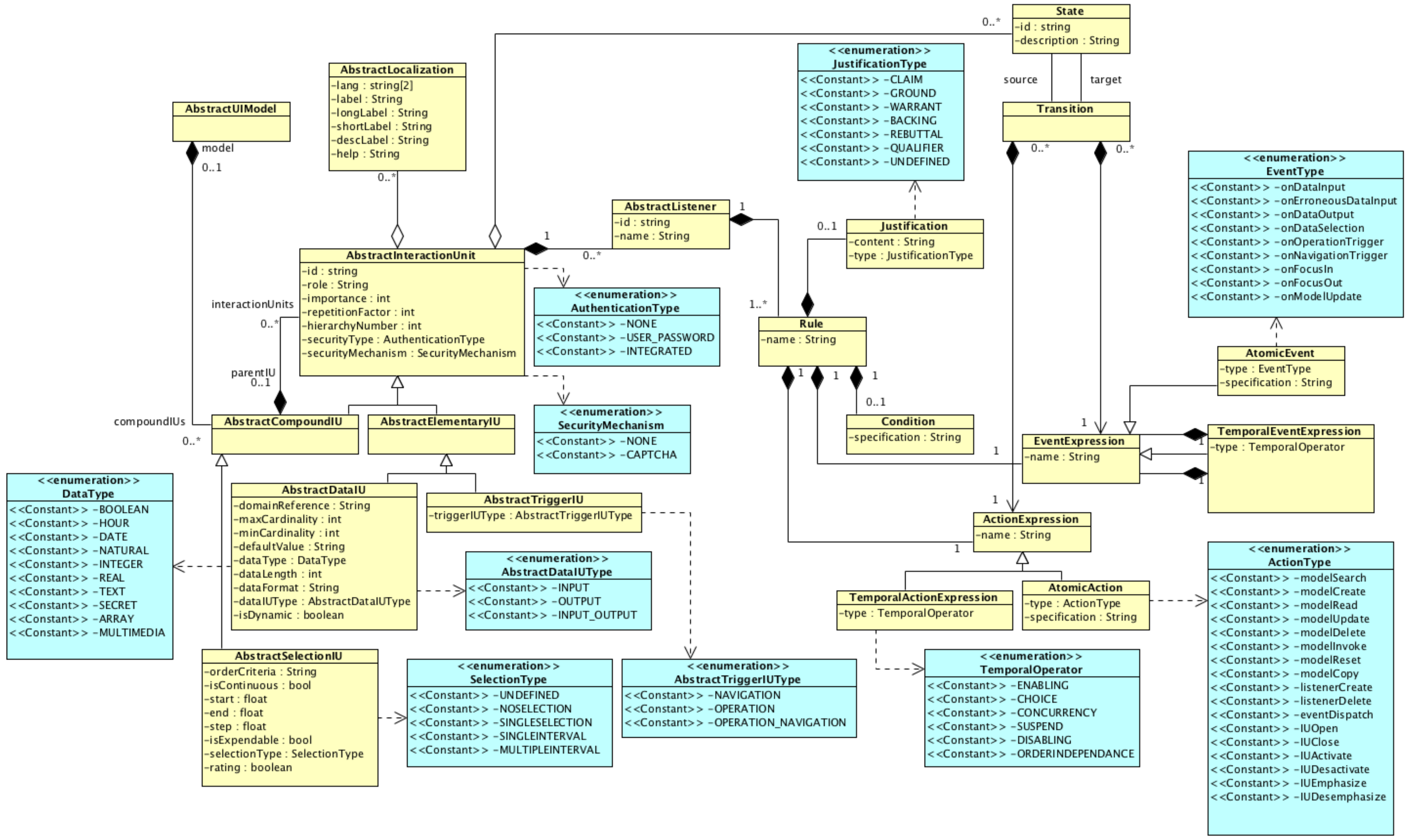

Figure 5 UsiXML

- **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.
- **State:** possible state of an AbstractInteractorUnit (user defined)
- **Transition**: represents a transition from a source to a target State.
- **AbstractLocalization.** Describes text attributes for AbstractInteractionUnits. It is useful for internationalization.
- **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.
- **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.
- **AbstractTriggerIU.** Interaction unit allowing triggering an event.
- **AbstractListener.** Entity used to describe the behavior of the AbstractInteractionUnit by using Event-Condition-Action (ECA) rules.
- **Rule.** Represents a ECA rule.
- **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.
- **EventExpression.** Specifies a set of events with relationships between them. The events are represented by AtomicEvents and are related by TemporalEventExpressions.
- **Condition.** Logical test that, if satisfied or evaluates to true, causes the action to be carried out.
- **ActionExpression.** Specifies a set of actions with relationships between them. The actions are represented by AtomicActions and are related by TemporalActionExpressions

2.2 Main Modelling Constructs

The following is a list of the different concepts that have been identified in the meta-models described in the previous sections. For each one of them, we include a description together with a table where we specify whether a language has an equivalent concept and, if it is the case, provides the alias for identifying it.

Please note that the comparison considers the previous version of the ASFE-DL [D3.2.1] and, as discussed in the following sections, it provided the input for some modifications of the language.

Interactor: an interactor is a generic interaction object, which can be exploited by both the application and the user in order to communicate with each other. The categories usually identified for this kind of object are: edit, output, select, trigger. Some languages represents such categories as subclasses of the interactor concept (extension), others use attributes in order to qualify its role (comprehension).

ASFE-DL	MARIA	MINT-AIM	UseDM	UsiXML
✓ Abstract Interactor	✓ Interactor	✓ AIO	✓ Abstract Interaction object	✓ Abstract Interaction Unit

Interactor container: an interactor container allows the designer to group together a set of interactors (or inner containers) that are logically related.

ASFE-DL	MARIA	MINT-AIM	UseDM	UsiXML
✓ Abstract Relationship	✓ Interactor Composition	✓ AI Container	✓ Container	✓ Abstract Compound IU

Presentation: a presentation is a set of Interactors and Interactor containers that represent a single logical unit, which should be presented to the user at once.

ASFE-DL	MARIA	MINT-AIM	UseDM	UsiXML
✓ Abstract Interactor Unit	✓ Presentation	✗	✗	✗

Data Model: the data model defines the data types that are manipulated by the UI during the interaction with the user.

ASFE-DL	MARIA	MINT-AIM	UseDM	UsiXML
✓ Data Model	✓ Data Model	✗	✗	✓ Domain Model (external to the UI model, referenced with the *domainReference* attribute of an AbstractDataIU)

Dialog: defines an ordered set of actions between the user and the system.

ASFE-DL	MARIA	MINT-AIM	UseDM	UsiXML
✓ Abstract Dialog Model	✓ Dialog Model	✓ Behaviour (specified with SCXML)	✓ Behaviour	✓ Rule

Event: asynchronous notification of a status change.

ASFE-DL	MARIA	MINT-AIM	UseDM	UsiXML
✓ Event	✓ Event	✓ Observation (bridge between interactors and behaviour)	✓ Transitions (which are event based)	✓ EventExpression + AtomicEvent

Actions: description of a list of actions and changes to be performed on different models in order to move from one UI state to another.

ASFE-DL	MARIA	MINT-AIM	UseDM	UsiXML
✗	✓ Event Handler	✓ Action part of a transition in the behaviour SCXML	✓ Transition (action part)	✓ ActionExpression + AtomicActions

3 ASFE-DL: Abstract Level

In this section we discuss the changes to the abstract level of the ASFE-DL. Starting from the UML diagram depicted in Figure 1, we modified the definition in order to better support the use cases of the project, and also according to the inputs and questions raised during the discussion in the W3C MBUI working group. The result is an enhanced version of the Abstract User Interface level, which constitutes the basis for the Concrete Desktop language we define in section 4.

In the following paragraph we describe the changes to the different parts of the models, which have been highlighted in the UML diagram in Figure 1 using different colours: sky-blue for the main structure of the interface, green for the interactor hierarchy, red for the classes that model the relationships between interactors and yellow for the classes that model the UI behaviour.

3.1 Main Structure

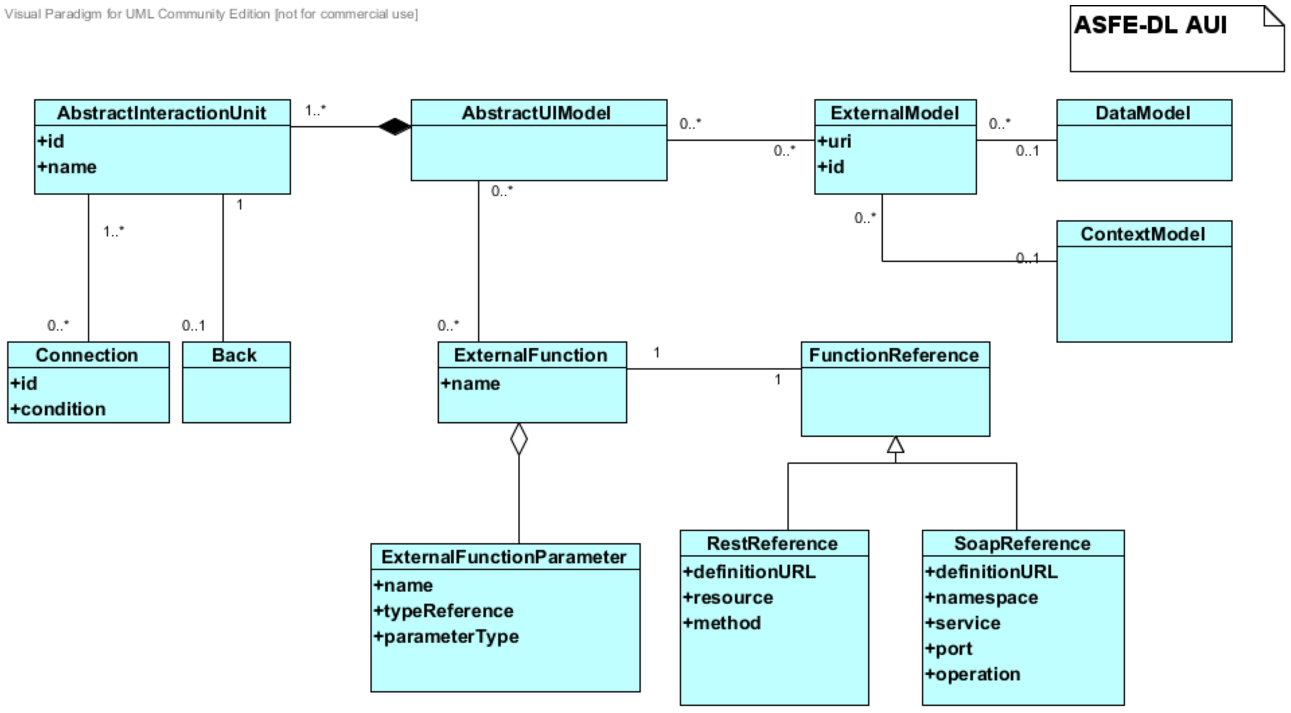

Figure 6: ASFE-DL AUI, main structure (updated version)

Figure 6 shows the UML class diagram of the main structure of the ASFE-DL Abstract User Interface. We introduced three major changes to its main, higher-level structure with respect to the previous version. Two of them allowed also to align the ASFE-DL with the AAL-DL for the parts that can be shared between the two languages. Indeed, both of them need to refer to some other models in order to define concepts that are exploited by the UI, but they are not directly related with the User Interface (or the adaptation rules in the case of the AAL-DL). Examples of such kind of models are the *DataModel* and the *ContextModel*, whose values influence both the rendering and the behaviour of a UI, but that can be managed separately with respect to the UI model.

We introduced the class *ExternalModel* in order to define references to models that are needed by the AUI, but that are not defined into the AUI itself. An external model reference consists of a unique *id* which is used in the AUI definition for referencing the external model, and by an *URI* that is used for locate the actual external model definition.

The second alignment with the AAL-DL language has been the introduction of the *ExternalFunction*s, which defines a list of functionalities, external to the modelled application, which are exploited inside the model. Since we are defining a language for describing Service Front Ends, the *ExternalFunction* element is a convenient way for referencing Web Services: in the AUI we define the signature of the service operation through the instances of the *ExternalFunctionParameter* class. It is possible to specify a parameter name, give a reference to its type (the *typeReference* attribute) and to specify if it is an input or an output parameter (*parameterType*). Such signature is independent from the underling service technology and it can be mapped to the two different types of Web Services supported at the moment: SOAP-based or REST-based. In order to unambiguously identify the service operation, an external function can be linked to two different types of references, which are represented by the *SoapReference* and the *RestReference*. Each one contains a set of attributes that allows the identification of the operation according to the represented technology.

The last major change to be reported for the main structure is the introduction of the *Back* class, a new type of connection. A connection in ASFE-DL declares that from a given *AbstractInteractionUnit* (AIU) it is possible to reach another AIU. A *Back* instance connects the current AIU with the previously visited one. If the current AIU can be reached from more than one AIUs, the target of the back connection is different according to the user's navigation. For instance, suppose that the current AIU is *c* and it can be reached from both the *a* and *b* AIUs. If the user has visited *c* from *a*, then the back connection target is *a*, while if the user has visited *ci* from *b*, the back connection target is *b*.

Minor changes have been the introduction of an attribute in the *AbstractUIModel* for maintaining the current rendered presentation. In addition, we added also a *role* attribute to the AIUs, in order to allow the designer to specify a semantic tag for them. The semantics of the tag is completely up to the interface designer, therefore we do not assign a unique way for interpreting the values. Such kind of attribute has been proved useful from the IDEAL2 [Cantera2010] experience, where it is possible to define groups of presentations related to the same task through this attribute.

3.2 Relationship between interactors

The modelling elements that define relationship between the interactors did not have major changes. They are still represented by the *AbstractRelationship* class, which is the base for all the different relations between the UI components. We have five different types of abstract relationships: the *Ordering* that defines an ordering relationship between the contained elements, the *Hierarchy* that defines a set of interactors where different levels of importance can be identified, the *Repetition* that defines a template for a list of interactor that have to be repeated in order to represent a dynamic list of items, the *Dependency* that models a dependency between one interactor/interactor group and other N interactors/interactors group and the *Grouping* that represents a generic group of interactors which share a logical relationships.

In order to better specify the logical relationship among the elements of a *Grouping*, we introduced a *role* attribute, which marks semantically the rationale for creating a group for a given set of UI elements. The possible values of the role attribute are the following:

- *None*, the default value for the attribute, it is used for a generic (not better specified) grouping rationale.
- *Article* marks the group as self-contained, which means that it should be possible to distribute the inner contents independently from its surrounding context
- *Aside* defines a content that contains a digression or a more detailed description of the content around it
- *HeaderGroup* defines the headings of a section or an entire document, the larger is the main one, while the others are sub-headings
- *Navigation* defines a group of elements that allow the navigation of the application contents
- *Section* defines a part of the application contents that has a particular theme and typically includes a heading.
- *Menu* defines a group that contains a list of commands
- *Geo* defines a group that contains information on how to locate places in the real world.

3.3 Interactor Hiearchy

The interactor hierarchy part of the ASFE-DL model did not have major changes. It still contain a superclass for each type of interactor (*AbstractInteractor*) which is refined into different classes, according to the interaction semantics:

- *Selection* for the interactors that allow the user to select one or more values from a predefined set of choices.
- *Edit* for the interactors that allow the input of manually edited values
- *OnlyOutput* for the interactors that show information to the user
- *Control* for the interactors that allows the trigger of a functionality or the navigation inside the application.

One minor change we applied was the modelling of the choice options for the *Selection* interactors at the abstract level. They are represented by object of the *ChoiceElement* class, which contains one attribute for referencing the value of the option.

3.4 Behaviour

The behaviour part of the ASFE-DL schema is the one that we refined more, in order to have a precise way to describe the dynamic changes in the UI according to the interaction with the user. The purpose of the refinement we are going to discuss is substantially shared with the Action part of the AAL-DL format for describing adaptation rules (see [D3.3.1]), which express the changed that have to be applied to the UI (and also to other models) in order to react to context changes. Therefore, we decided to exploit the same modelling constructs in both languages.

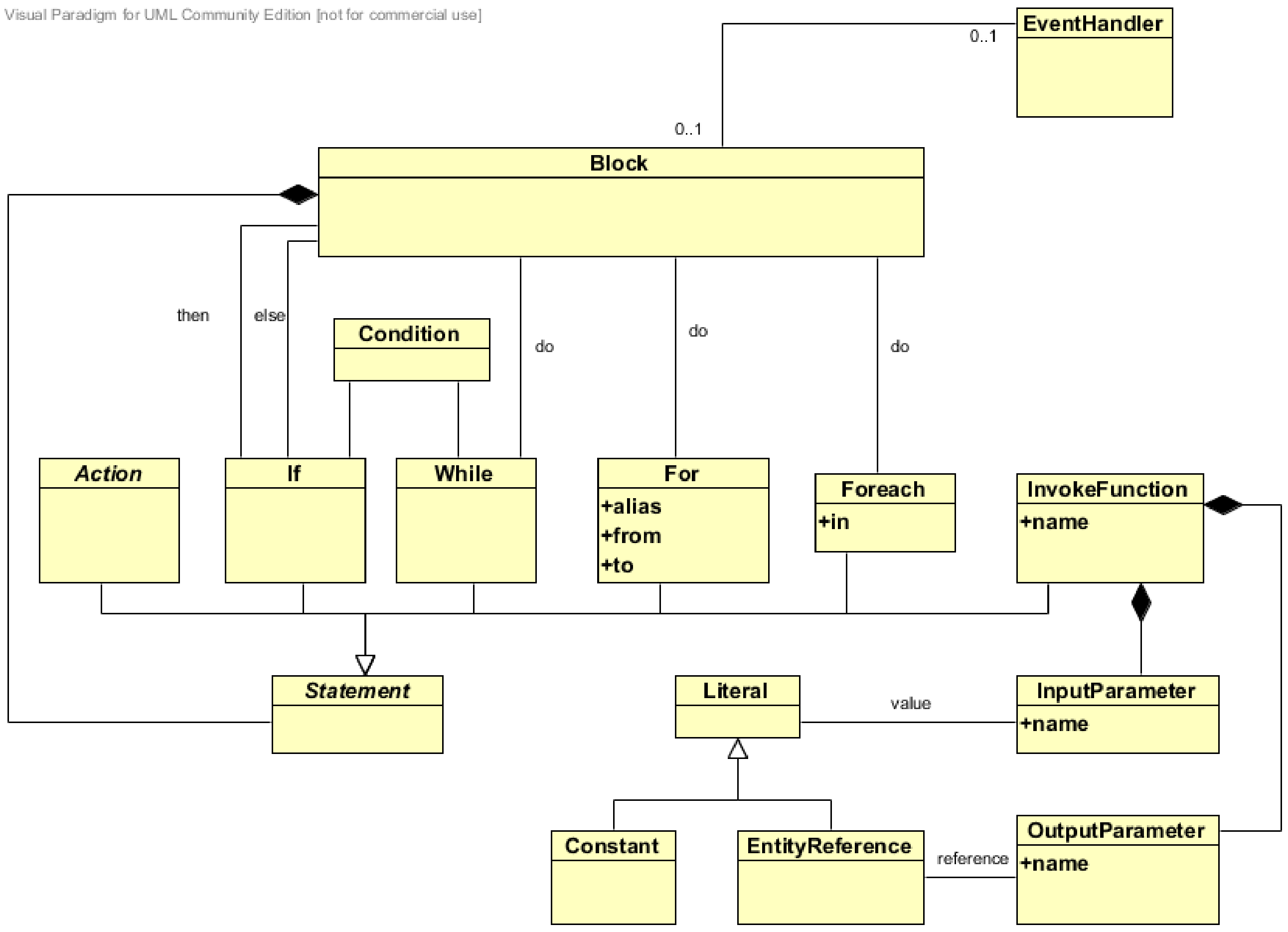

Figure 7: Statements class diagram

In order to have a complete model for defining the dynamic behaviour, we introduced the classic set of *Statements* of the imperative programming languages (see Figure 7). Therefore, an *EventHandler* is simply a *Block*, which represents a collection of *Statements*. The class names for such statements are self-explaining: *If*, *While*, *For*, *Foreach*. In addition, we introduced a statement that allow the UI to invoke external functionalities, declared through the *ExternalFunction* instances (see Figure 6). The *InvokeFunction* references the *name* of the corresponding *ExternalFunction*, specifying the binding between the formal and the actual parameters through the instances of *InputParameter* and *OutputParameter* class.

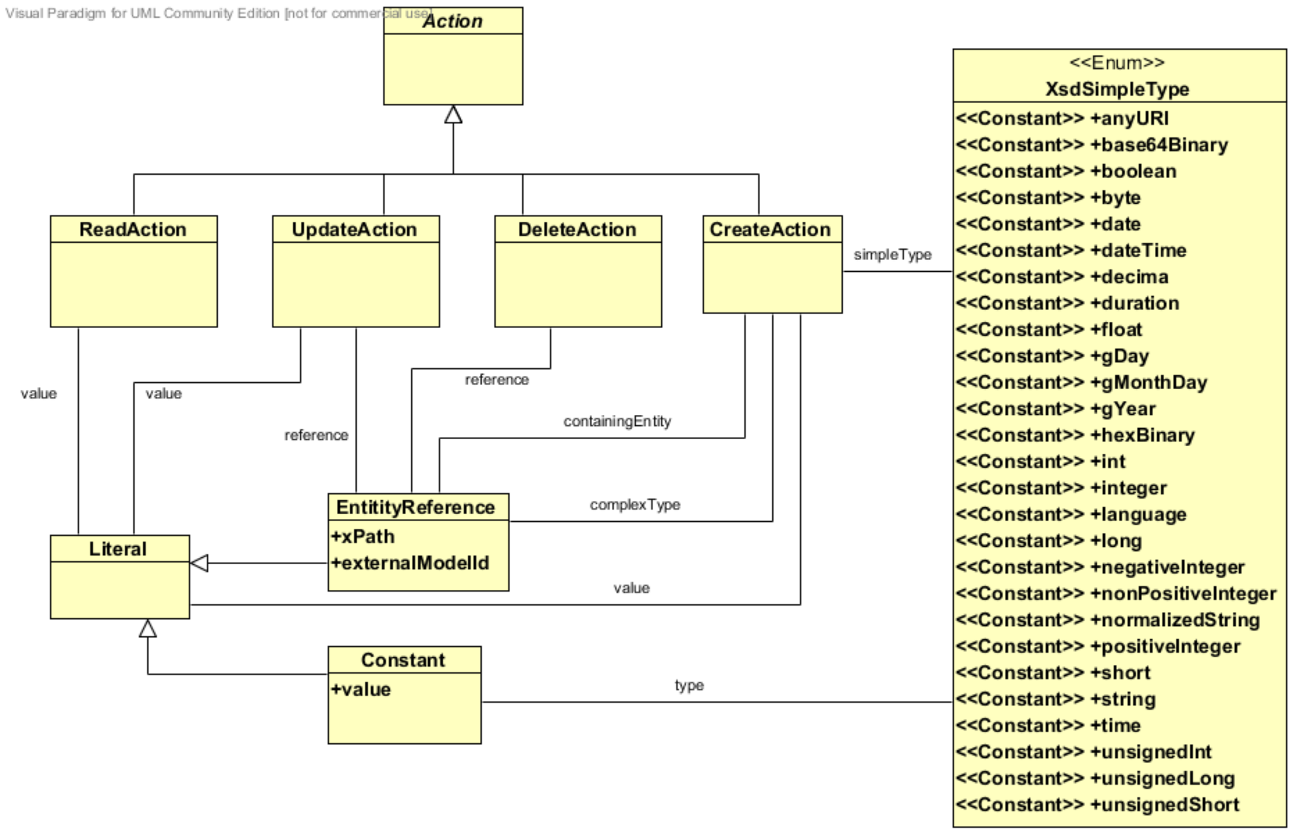

Figure 8: Action class diagram

Figure 8 shows the basic set of actions that allow the definition of the dynamic changes to the model state. First of all, we need to define how to reference *Constant* values (defined through a *value* and a simple *type*) or variables, represented as *EntityReference*s. An *EntityReference* can be used in order to point to a value which can be dynamically changed and also for referencing type definitions. In the former case, the external model identified by the attribute *externalModelId* references an XML document, in the latter case, the external model points to an XSD. Both *Constant*s and *EntityReferences* are subclasses of *Literal*, which represent a generic value.

The possible types of basic actions (represented by the abstract class *Action*) are defined by the four classic functions of the persistent storage: *Create*, *Read*, *Update* and *Delete*.

- **Create**. The action creates a novel model entity, optionally initializing its value. The attribute *containingEntity* provides the reference to the model entity that will contain the new one, and it is defined by an instance of the *EntityReference* class. The attribute *simpleType* and *complexType* define the type of the new attribute, either simple (with an instance of the *XsdSimpleType* enumeration) or complex (with an instance of the *EntityReference* class).
- **Read**. The action reads a model entity or a constant *value*, represented by an instance of the *Literal* subclasses.
- **Update**. The action changes the current value of a model entity, defined through the *reference* attribute (of type *EntityReference*). The new value for the entity reference is defined by the *value* attribute, which is an instance of the *Literal* subclass.
- **Delete.** The action deletes an element from the model. The *reference* to the element to be deleted is defined through an instance of the class *EntityReference*.

4 ASFE-DL: Concrete Desktop

The definition of a Concrete Language consists mainly on the refinement of the abstract classes with the elements that can be used in the specific platform in order to build the UI.

We started from the most common one, the Desktop, and we consider in the following sections the different Abstract Interactors and Abstract Relationship refinements that are possible to use in this platform.

4.1 Styles

Before describing the different refinements, we provide here the list of the different style properties that can be associated to the different concrete interactors. We group the properties in different categories and, for each modelling element, we specify which groups of properties are supported, without reporting all the attributes in the UML diagram.

In order to describe the visualization properties of an element, we exploit the W3C definitions of the dimensions of a box, depicted in Figure 9.

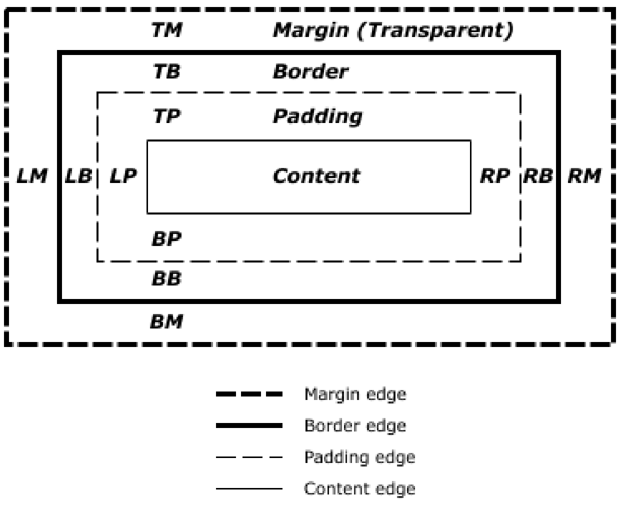

Figure 9: Box dimensions

The following table lists the style properties:

Group	Name	Definition
Background	background-attachment	The position of the background image with respect to the viewport: *fixed* or *scroll*
	background-color	The background color of an element
	background-image	The background image of an element
	background-position	The initial position of a background image with respect to the element
	background-repeat	Specifies whether the background image has to be repeated and how (on the vertical axis, on the horizontal axis or both axes).
	background	Specifies all the background properties in a single line, in the following order: *color, image, repeat, attachment, position*
Table	caption-side	Specifies whether the table caption should be put at the top or the bottom of the table

Group	Name	Definition
	border-collapse	Specifies whether the inner and the outer border of a table should be *collapse*d or *separate*d.
	border-spacing	Specifies the distance between the border of two adjacent cells
Border	border-color	Specifies the colour of all the borders of an element, using the same one for the *top*, *right*, *bottom*, *left* part or specifying each one with the given order.
	border-style	Defines the style of the line used for drawing the element border (e.g. solid, dotted, dashed, inset, outset etc.)
	border-top border-right border-bottom border-left	Defines all the properties of the different borders in the following order: *top*, *right*, *bottom*, *left*
	border-top-color border-right-color border-bottom-color border-left-color	Defines the border colour respectively for the *top, right, bottom* and *left*.
	border-top-style border-right-style border-bottom-style border-left-style	Defines the style of the line (e.g. solid, dotted, dashed, inset, outset etc.) used for drawing respectively the *top*, *right*, *bottom* and *left* border.
	border-top-width border-right-width border-bottom-width border-left-width	Defines the width respectively for the *top*, *right*, *bottom* and *left* border
	border-width	Specifies the width of all the borders of an element, using the same one for the *top*, *right*, *bottom*, *left* part or specifying each one with the given order.
	border	Specifies all the border properties in the following order: *width*, *style* and *color*
Position	bottom	Defines the distance between the element bottom margin and the bottom of the containing element.
	left	Defines the distance between the element left margin and the left edge of the containing element
	clear	This property indicates which sides of an element's element (*none*, *left*, *right*, *both*) may not be adjacent to an earlier floating box
	float	This property specifies that the box of an element should float to the *left*, to the *right* or *none*.
	position	Specifies the position of the element is *fixed*, *relative* to the containing element or *absolute*.
	right	Defines the distance between the element right margin and the right edge of the containing element
	top	Defines the distance between the element top margin and the top edge of the containing element

Group	Name	Definition
Text	color	Specifies the text colour
	direction	Specifies the writing direction for the element (*ltr* from left to right and *rtl* for right to left).
	font-family	Specifies the font family of the contained text
	font-size	Specifies the size of the font for the element text
	font-style	Specifies the style for the element text (*normal*, *italic*, *oblique*)
	font-variant	Specifies whether the text should be rendered in small capital letters (*small-caps*) or normally (*normal*)
	font-weight	Specifies the weight of the font, with increasing values of darkness or simply specifying that is should be rendered in bold.
	font	Specifies all the properties for the font of a text: *style*, *variant*, *weight*, *size*, *line-height*, *family*.
	line-height	Specifies the height of line of text
	text-align	Specifies whether the text should be aligned to the *left*, *center* or *right*.
	text-decoration	Specifies the text decoration *underline*, *overline*, *line-through*, *blink*.
	text-indent	Specifies the width of the text indentation
	text-transform	Specifies a specific transformation for the text: *capitalize*, *uppercase*, *lowercase*, *none*
	'vertical-align'	Specifies the vertical alignment of a text*: baseline* , *sub*, *super*, *top*, *text-top*, *middle*, *bottom*, *text-bottom*.
Display	cursor	Specifies the cursor to be displayed for the pointing device
	display	Specifies how the box of the element should be displayed (*inline*, *block*, *none* ect.)
	height	Specifies the height of the content of an element
	max-height	Specifies the maximum height of the element box
	max-width	Specifies the maximum width of the element box
	min-height	Specifies the minimum height of the element box
	min-width	Specifies the minimum width of the element box
	overflow	Specifies whether the content that overflows the box should be visible, or it may be scrolled.
	width	Specifies the width of an element
	z-index	Specifies the z-index of an element
List	list-style-image	Specifies an image for a list item marker
	list-style-position	Specifies whether the list item marker is inside the list element box or not.

Group	Name	Definition
	list-style-type	Specifies the list item marker from a set of predefined values (e.g. *disc, square, lower-roman, upper-roman* etc.).
	list-style	Specifies all the list attributes in the following order: type, position, image
Margin	margin-right margin-left' margin-top margin-bottom	Specifies respectively the width of the margin *right, left, top* and *bottom*.
	margin	Specifies a single value for all margins or each single value in the following order: *top, right, bottom, left*
Padding	padding-top padding-right padding-bottom padding-left	Specifies respectively the width of the padding *right, left, top* and *bottom*.
	padding	Specifies a single value for all margins or each single value in the following order: *top, right, bottom, left*

4.2 Input device events

The input devices provided by the Desktop platform are the pointer and the keyboard. Each model element may receive notifications about the status of pointer position and keyboard or mouse buttons. Therefore, in the ASFE-DL Concrete Desktop model we introduce two classes that represent such events: the *MouseEvent* and the *KeyboardEvent*. The former notifies about the position of the pointer (the *x* and *y* attribute) and the status of the buttons (*button1*, *button2*, *button3*) and the *scrollAmount* for the scrolling wheel. The latter specifies which button of the keyboard has been pressed (*keyCode*) and which keyboard modifier has been pressed (any combination of *ctrl*, *alt* and *shift*), if any.

Both classes have subtypes define a specific event and precisely:

- **Mouse Event**
  - *Mouse Enter*, raised when the mouse pointer enters into the area of an element
  - *Mouse Over*, raised when the mouse pointer moves inside the area of an element
  - *Mouse Leave*, Raised when the mouse pointer leaves the area of an element
  - *Mouse Click* raised when the user clicks a mouse button
  - *Mouse Double Click* raised when the user double clicks a mouse button
  - *Mouse Down* raised when a mouse button is pressed
  - *Mouse Up* raised when a mouse button is released.
- **Keyboard Event**
  - *Key Down*, raised when a key is pressed
  - *Key Up*, raised when a key is released
  - *Key Pressed*, a combination of key down and up.

Figure 10 shows the UML class diagram for the device-related events, the blue classes are related to the concrete platform.

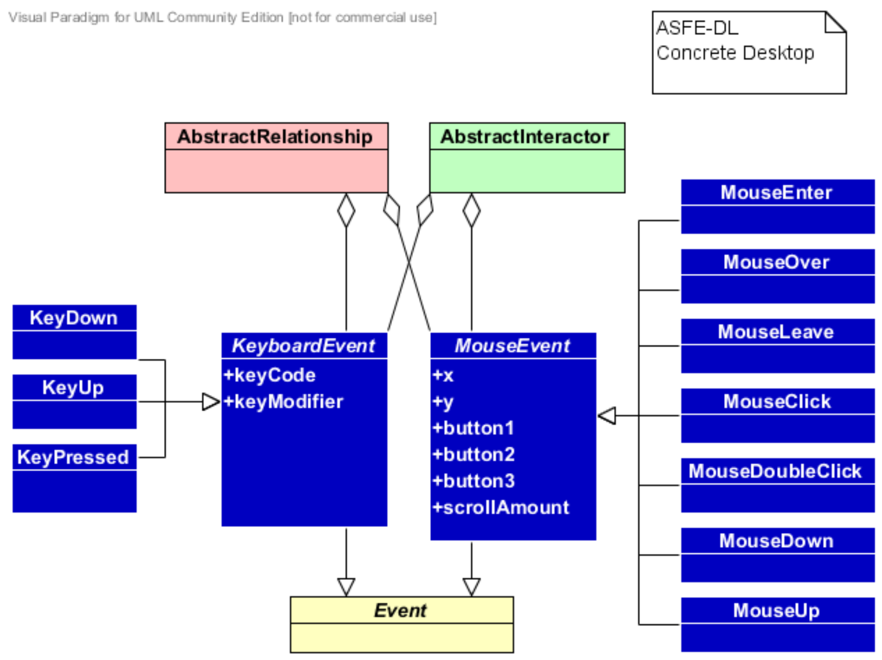

Figure 10: Concrete Desktop input device events

4.3 Concrete Desktop Interactors

4.3.1 Control

The concrete refinements for the *Activator* and the *Navigator* class are the *Button*, *Link* and *Image Map*. While the first two elements contain a combination of *Text* and *Images* that provide their label, the *Image Map* contains an *Image* and a set of *Polygon*s which represent the clickable areas. The Control hierarchy is represented in Figure 11. The possible styles properties group for all the elements are: background, border, position, text, display, margin and padding.

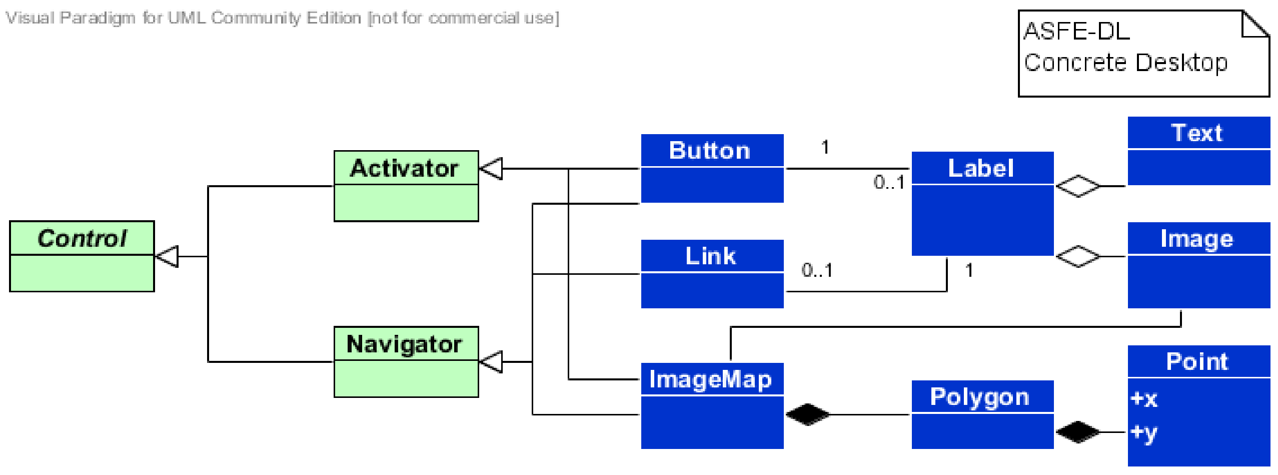

Figure 11: Control Interactors refinement

4.3.2 Edit

The following is the list of the concrete refinements of the Edit Interactors

- **Text Field,** a field for entering text on a single row.
- **Text Area,** a field for entering text on multiple lines
- **Spin Box**, a field for entering numerical values buttons for increasing and decreasing the values
- **Track Bar,** a bar with a draggable knob for entering values in a given range
- **Map**, a map control for entering positions.

All of them may contain a list of texts and images that represents the element labels.

The possible styles property groups for all the elements are: background, border, position, text, display, margin and padding.

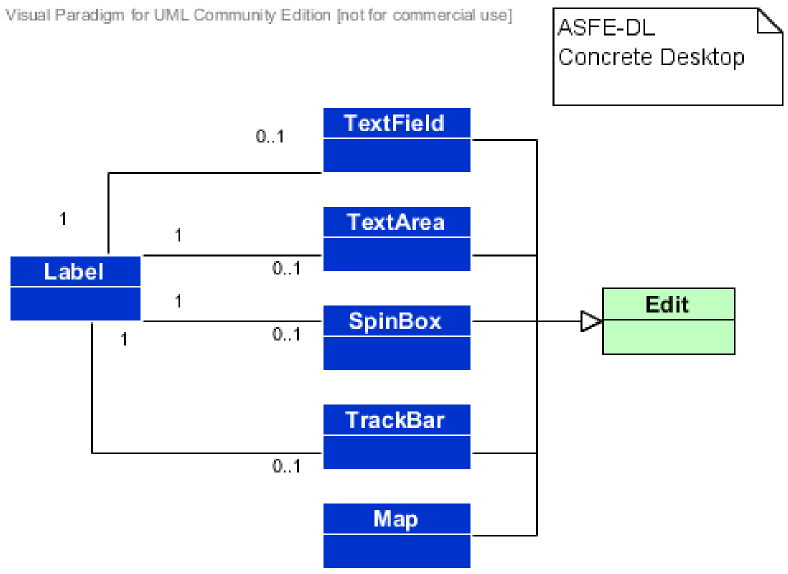

Figure 12: Edit refinement

4.3.3 Only output

The following is the list of the concrete refinements of the Only Output interactors, depicted in Figure 13.

- **Text**, which represent a textual content (provided by the *content* attribute). It contains a *role* attribute that defines which specific part of text is represented by the current element (e.g. an heading from 1 to 6, paragraph, strong, emphasis, figure caption etc.) The possible style property groups are: background, border, position, text, display margin, padding.
- **Image,** which represent an image content. It contains attributes for specifying the source (*url*) and the alternative content (*alt*). The possible style property groups are: background, border, position, display, margin and padding.
- **Table,** which represent a set of contents displayed in a tabular form. The possible style property groups are: background, border, table, position, text, display, margin, padding.
- **Audio,** which represents an audio content, together with the controls for playing, pausing, stopping and change the position in the stream. It contains an attribute for providing the alternative content (*alt*), a flag for indicating if the playback starts automatically (*autoplay*) and another one for specifying whether the playback controls should be visible or not (*controls*). The possible style property groups are: background, border, position, display, margin, padding.
- **Video,** which represents a video content, together with the controls for playing, pausing, stopping and change the position in the stream. It contains an attribute for providing the alternative content (*alt*), a flag for indicating if the playback starts automatically (*autoplay*) and another one for specifying whether the playback controls should be visible or not (*controls*). The possible style property groups are: background, border, position, display, margin, padding.
- **Progress Bar**, which represent a bar that can be displayed in order to show the progress of an operation that may take a long time. It contains an attribute indicating whether the waiting is *indefinite* or if the bar indicates a specified *value* (which is also represented with an attribute). The element contains the *Bar* sub-element which represents the bar itself and contains attributes for indicating whether the textual value should be written on the bar or not. The possible style property groups are: background, border, table, position, text, display, margin, padding. It is possible to associate them to both the whole element for defining the appearance of the non-completed part of the bar, and to the *Bar* sub element, which defines the appearance of the completed part.

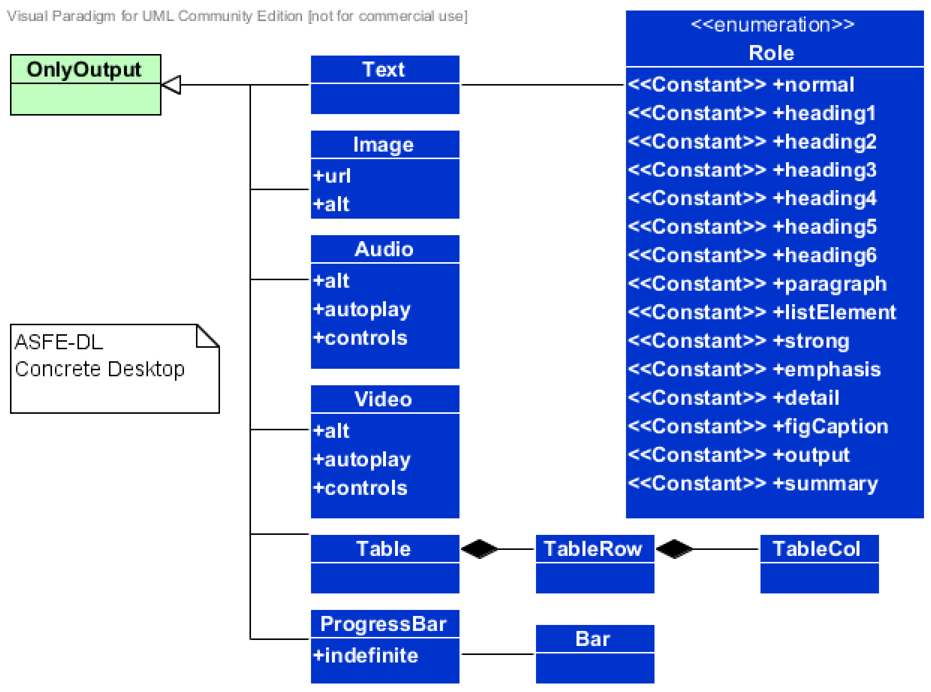

Figure 13: Only Output refinement

4.3.4 Selection

The following is the list of refinements for the *Single Choice* Selection Interactors, depicted in Figure 14.

- **Radio Button**, which shows the choice elements together with an associated button for the selection
- **List Box**, which shows the choice elements grouped inside a box, where it is possible to highlight one of them. It contains the *row* attribute that specifies how many options are visible at once, without scrolling them.
- **Drop Down List,** which shows the currently selected object together with a button that allows the visualization of the other options.
- **Image Map**, which allows the graphical selection of an area in an image.
- **Calendar**, which allows the selection of a date.

The *Multiple Choice* Selection Interactors instead can be refined as follows:

- **Check Box,** which shows the choice elements together with a check for selecting them.
- **List Box,** which is the same with respect to the previous one, with the only difference that more than an option can be highlighted (specified with the *multiple* attribute).

All the elements can be associated to the following style property groups: background, border, position, text, display, margin, padding.

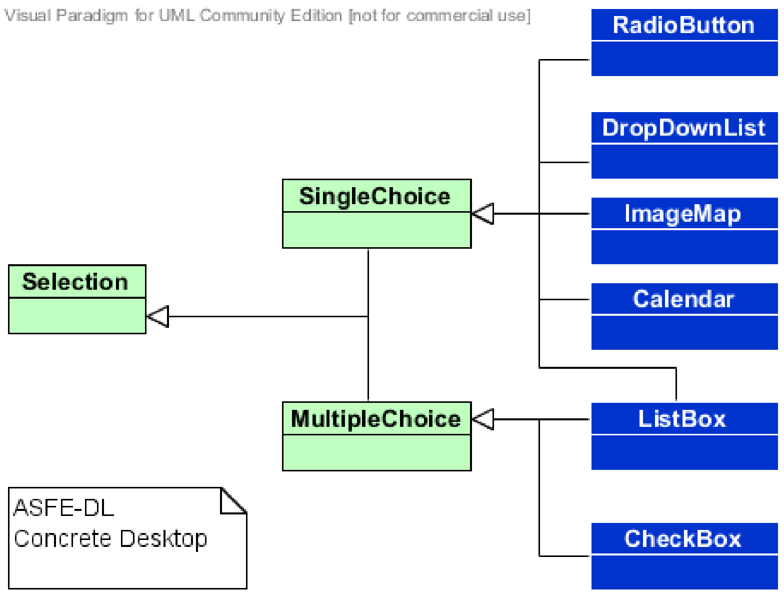

Figure 14: Selection refinements

4.4 Concrete Desktop Relations

Figure 15 shows the UML class diagram of the concrete refinements of the *AbstractRelation* classes in the concrete desktop platform.

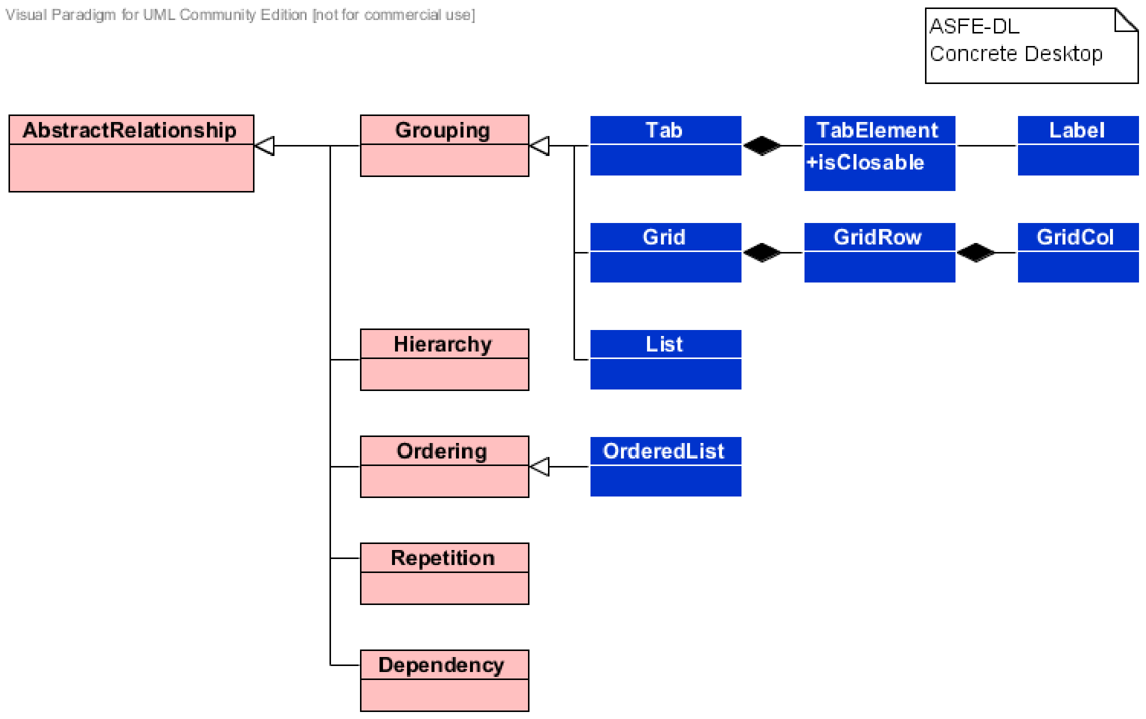

Figure 15: Relationship refinements

4.4.1 Grouping

The *Grouping* class can be directly used also in the Concrete Desktop platform. In addition, it is possible to use the *Grid*, the *Tab* and the *List* refinements that specify a different layout for the group. The *Grid* allows the designer to display the different contents in a table-like layout, without using a table. Therefore, in the grid specification, there are no headings for the rows and columns. The *Tab* element allows to display only one of the different *TabElement* contents at a given time. Each *TabElement* has an associated label that is displayed in the upper, lower, left or right part of the element. It is possible to change the currently selected tab clicking on its label. The *List* displays its inner contents using a bullet list. The associated style property groups for all the elements are: background, border, position, text, display, margin and padding. The *List* class has in addition the list group.

4.4.2 Hierarchy

The *Hierarchy* class can be directly used in a Desktop CUI. The associated style property groups are: background, border, position, text, display, margin and padding.

4.4.3 Ordering

The *Ordering* Relation can be used directly or through its *OrderedList* subclass. The latter represent an ordered list of items. Both base and subclass are associated to the background, border, position, text display, margin and padding style groups. In addition, the *OrderedList* class is associated also to the list group.

4.4.4 Repetition

The *Repetition* element can be used directly into a desktop CUI and it is associated to the background, border, position, text, display, margin and padding style groups.

4.4.5 Dependency

The *Dependency* element can be used directly into a desktop CUI and it is associated to the background, border, position, text, display, margin and padding style groups.

5 Conclusions

5.1 Summary

In this deliverable we described the progress on the definition of a reference language for the description of SFE that can be adapted to the context. In this regard, we presented a survey on the various submissions to the W3C Model Based User Interfaces Working Group, which has the mission to define a standard for an Abstract User Interface Language. From these submissions, we extracted the common concepts and we compared them to the ASFE-DL language presented in the first version of this deliverable.

After that, we described all the changes that have been applied to the previous definition of the language, which affected its main structure, the relationships between interactors, the interactor hierarchy and the definition of the UI behaviour.

Finally, we described the refinement of the ASFE-DL Abstract level for describing concrete desktop interfaces.

5.2 Future Work

Since we described the intermediate version of the language, the work in this task (T3.2) needs to be continued for the remaining part of the project. In particular, we still need to address the specification of the language for platforms different from the desktop, and in particular those that do not provide a graphical output. In addition, the results of the work in this task have to be coordinated with the definition of the Advanced Adaptation Logic Description Language, which is the output of T3.3. As we already mentioned in this deliverable, and in particular in section 3.4, these tasks may provide inputs to each other in order to reach a common definition of the shared parts.

Finally, the prototypes will take advantage of the new definitions in order to create a more integrated solution to the context adaptation problem.

6 References

[D3.2.1] Deliverable 3.2.1: ASFE-DL Semantics, Syntaxes and Stylistics (R1) , The Serenoa Consortium.

[D3.3.1] Deliverable 3.3.1: AAL-DL Semantics, Syntaxes and Stylistics (R1) , The Serenoa Consortium

[Calvary2002] The CAMELEON Reference Framework, G. Calvary, J. Coutaz, D. Thevenin, L. Bouillon, M. Florins, Q. Limbourg, N. Souchon, J. Vanderdonckt, L.Marucci, F.Paternò, and C.Santoro, CAMELEON Project, September 2002.

[Cantera2010] IDEAL2 Core language. José M. Cantera, C. Rodriguez, José L. Díaz.MyMobileWeb Working Draft, 31 December 2010. Available athttp://files.morfeo-project.org/mymobileweb/public/specs/ideal2/ideal2-20101231/. Latest version:https://files.morfeo-project.org/mymobileweb/public/specs/ideal2.

[Feuerstack2011]Sebastian Feuerstack, Ednaldo Pizzolato; Building Multimodal Interfaces out of Executable, Model-based Interactors and Mappings; HCI International 2011; 14th International Conference on Human-Computer Interaction; J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2011, LNCS 6761, pp. 221—228. Springer, Heidelberg (2011), 9-14 July 2011, Hilton Orlando Bonnet Creek, Orlando, Florida, USA.

[Paternò2009] F. Paternò, C. Santoro, L.D. Spano, "MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment", ACM Transactions on Computer-Human Interaction, Vol.16, N.4, November 2009, pp.19:1-19:30, ACM Press.

[Seissler2011] Seissler, M., Breiner, K., Meixner, G.: Towards Pattern-Driven Engineering of Run-Time Adaptive User Interfaces for Smart Production Environments. Proceedings of the 14th International Conference on Human-Computer Interaction. Springer (2011)

[UsiXML2011] UCL(ed), The UsiXML Consortium (2011), UsiXML Deliverable D1.3: UsiXML Definition,15 NOV 2011.

Acknowledgements

- TELEFÓNICA INVESTIGACIÓN Y DESARROLLO, http://www.tid.es
- UNIVERSITE CATHOLIQUE DE LOUVAIN, http://www.uclouvain.be
- ISTI, http://giove.isti.cnr.it
- SAP AG, http://www.sap.com
- GEIE ERCIM, http://www.ercim.eu
- W4, http://w4global.com
- FUNDACION CTIC http://www.fundacionctic.org

Glossary

- http://www.serenoa-fp7.eu/glossary-of-terms