MARIA (Model-based lAnguage foR Interactive Applications) W3C Working Group Submission 3 February 2012

Editors:
Fabio Paternò, ISTI-CNR
Carmen Santoro, ISTI-CNR
Lucio Davide Spano, ISTI-CNR

Abstract

This is a submission to the W3C Model-Based UI Working Group and describes a metamodel and XML format for defining abstract user interfaces and two concrete refinements: desktop and vocal.

1. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

2. Introduction

This section is non-normative.
MARIA (Model-based language for Interactive Applications), is a universal, declarative, multiple abstraction-level, XML-based language for modelling interactive applications in ubiquitous environments. For designers of multi-device user interfaces, one advantage of using a multi-layer description for specifying UIs is that they do not have to learn all the details of the many possible implementation languages supported by the various devices, but they can reason in abstract terms without being tied to a particular UI modality or, even worse, implementation language. In this way,they can better focus on the semantics of the interaction, namely what the intended goal of the interaction is, regardless of the details and specificities of the particular environment considered.

3. Abstract User Interface

The Abstract User Interface (AUI) level describes a UI only through the semantics of the interaction, without referring to a particular device capability, interaction modality or implementation technology.

An AUI is composed by various Presentations that groups model elements, which are presented to the user at once. The model elements are of two types: *Interactor* or *Interactor Composition*. The former represents every type of user interaction object, the latter groups together elements that have a logical relationship. According to its semantics an interactor belongs to one the following subtypes:

- ***Selection***. Allows the user to select one or more values among the elements of a predefined list. It contains the selected value and the information about the list cardinality. According to

the number of values that can be selected, the interactor can be a *Single Choice* or a *Multiple Choice*.

- ***Edit***. Allows the user to manually edit the object represented by the interactor, which can be text (*Text Edit*), a number (*Numerical Edit*), a position (*Position Edit*) or a generic object (*Object Edit*).
- ***Control***. Allows the user to switch between presentations (*Navigator*) or to activate UI functionalities (*Activator*).
- ***Only output***. Represents information that is submitted to the user, not affected by user actions. It can be a text a Description that represents different types of media, an *Alarm*, a *Feedback* or a generic *Object*.

The different types of interactor-compositions are:

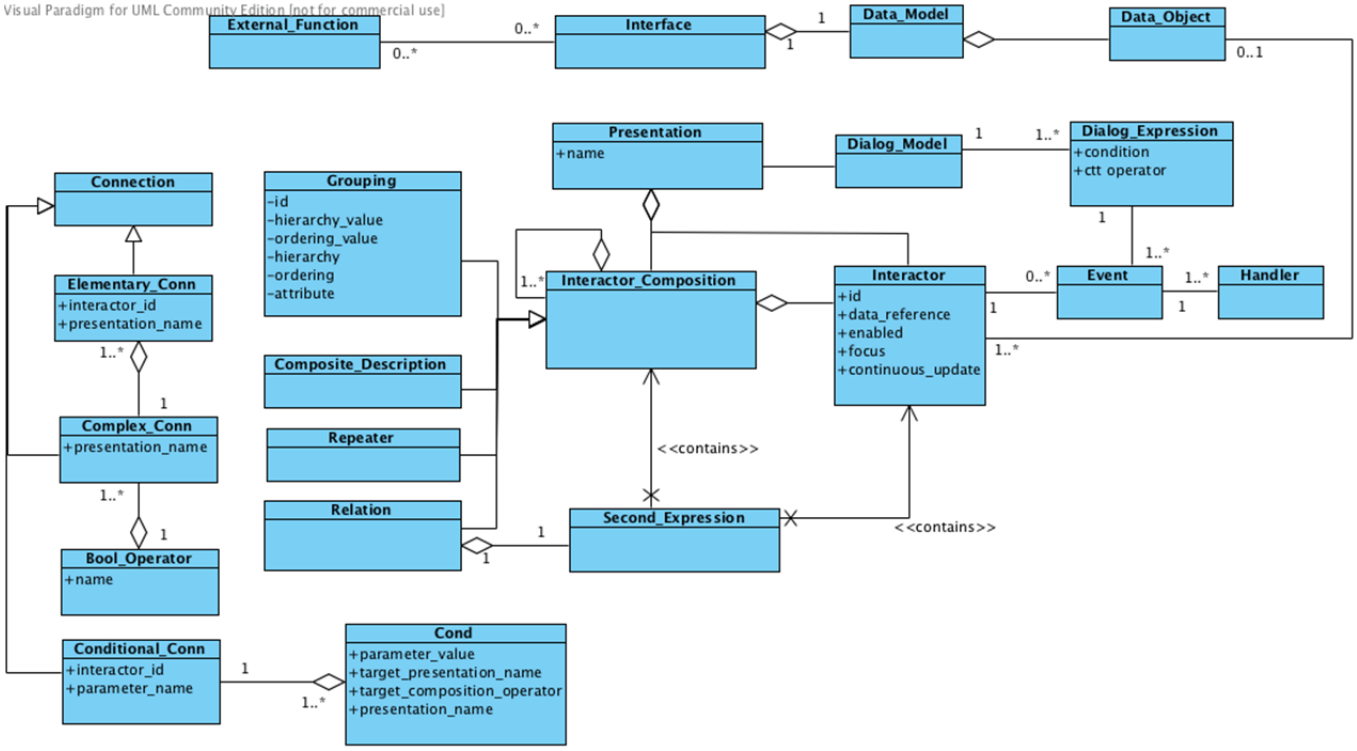
- ***Grouping***: a generic group of interactor elements.
- ***Relation***: a group where two or more elements are related to each other.
- ***Composite Description***: represents a group aimed to present contents through a mixture of *Description* and *Navigator* elements.
- ***Repeater*** which is used to repeat the content according to data retrieved from a generic data source

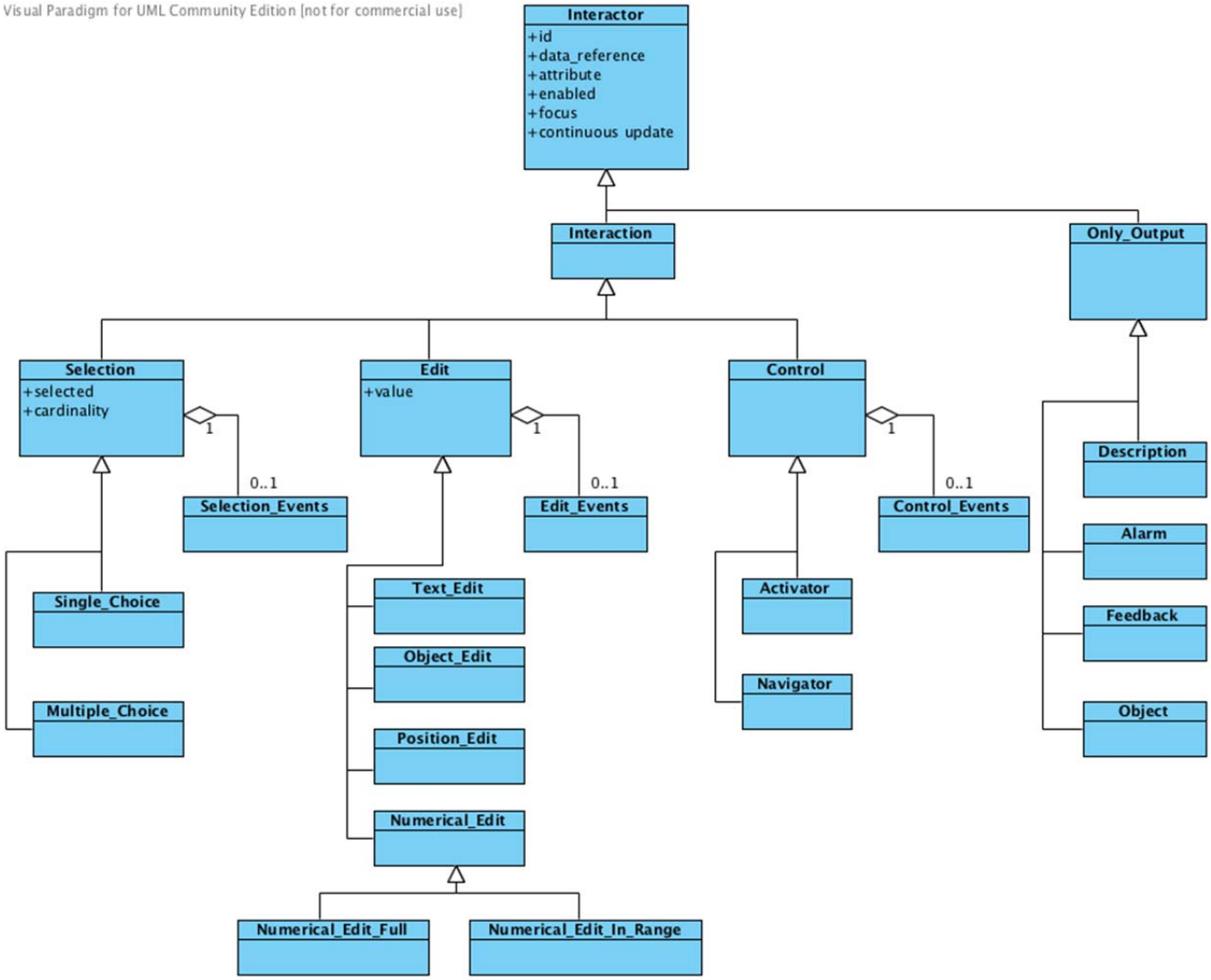
MARIA XML allows describing not only the presentation aspects but also the behaviour Data Model. The interface definition contains description of the data types that are manipulated by the user interface. The interactors can be bound with elements the data model, which means that, at runtime, modifying the state of an interactor will change also the value of the bound data element and vice-versa. This mechanism allows the modelling of correlation between UI elements, conditional layout, conditional connections between presentations, input values format. The data model is defined using the standard XML Schema Definition constructs.

- ***Generic Back End***. The interface definition contains a set of *External Functions* declarations, which represents functionalities exploited by the UI but implemented by a generic application back-end support (e.g. web services, code libraries, databases etc.). One declaration contains the signature of the external function that specifies its name and its input/output parameters.
- ***Event Model***. Each interactor definition has a number of associated events that allow the specification of UI reaction triggered by the user interaction. Two different classes of events have been identified: the Property Change Events that specify the value change of a property in the UI or in the data model (with an optional precondition), and the Activation Events that can be raised by activators and are intended to specify the execution of some application functionalities (e.g. invoking an external function).
- ***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 are connected using CTT operators in order to define their temporal relationships.
- ***Continuous update of fields***. It is possible to specify that a given field should be periodically updated invoking an external function.
- ***Dynamic Set of User Interface Elements***. The language contains constructs for specifying partial presentation updates (dynamically changing the content of entire groupings) and the possibility to specify a conditional navigation between presentations.

This set of new features allows having already at the abstract level a model of the user interface that is not tied to layout details, but it is complete enough for reasoning on how UI supports both the user interaction and the application back end.

Next Figures show the overall class diagram of the AUI meta-model and the possible interactor categories.





4. Desktop Concrete User Interface

A CUI meta-model for a given platform is an extension of the AUI meta-model, which means that all the entities in the AUI still exist in the CUI. The extensions add the platform-dependent information (but still implementation language independent) to the structure of the corresponding AUI model for the same application interface by either adding attributes or extending through an inheritance mechanism the existing entities for the specification of the possible concrete implementation of the abstract interactors.

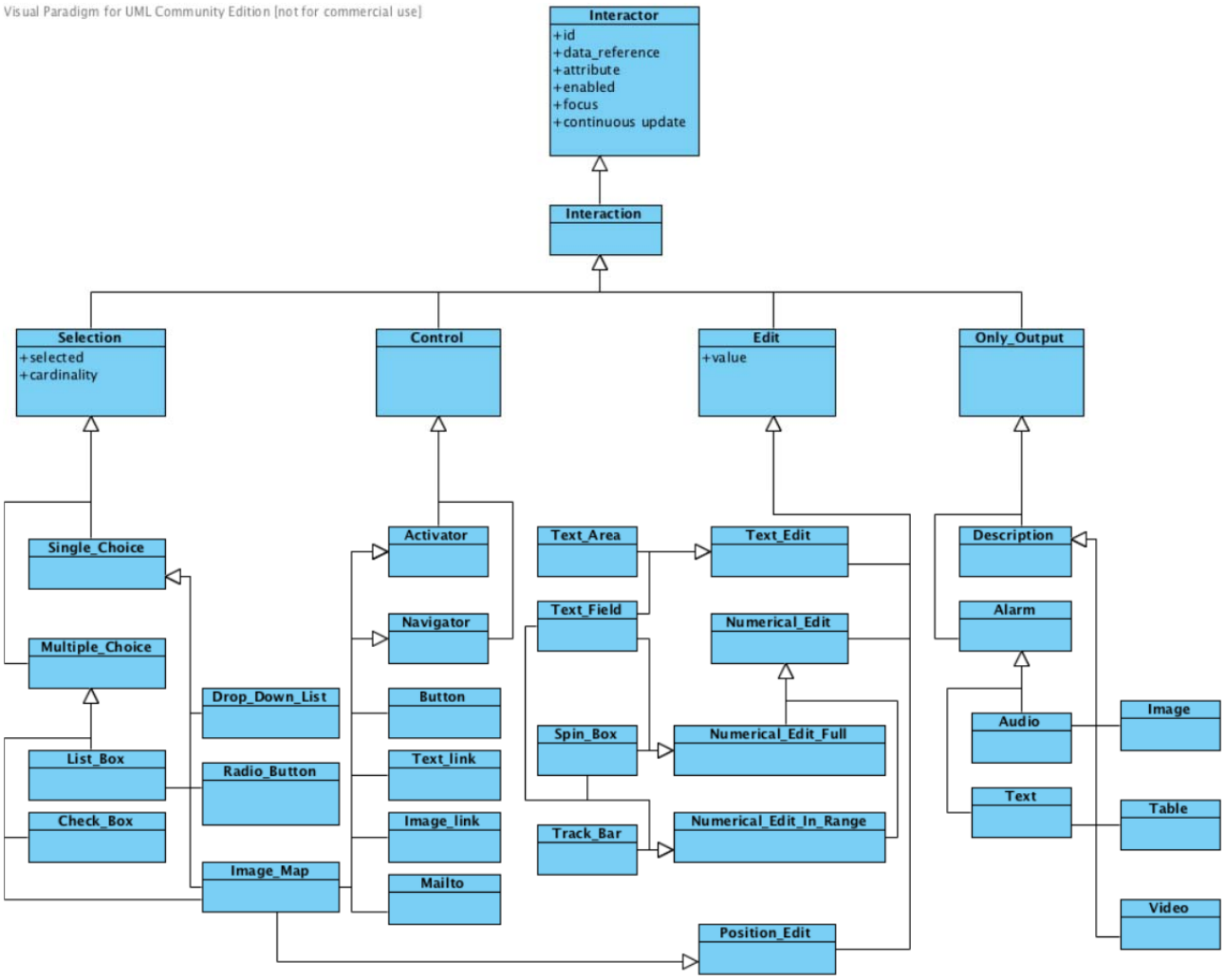
In this paragraph, we will introduce the extension to the AUI meta-model for the definition of the Graphical Desktop CUI meta-model. The existing elements with new attributes are:

Presentation: it contains the *presentation_setting* attribute, which contains information on the title, background (color or image) and the font used.

Grouping: it contains the *grouping_setting* attribute, which contains the information on the grouping display technique (*grid*, *fieldset*, *bullet*, *background color* or *image*) and if the elements are related with an ordering or hierarchy relation.

The classes which have been extended using the inheritance are the following:

- An *Activator* can be implemented as a *button*, a *text_link*, *image_link*, *image_map* (an image with the definition of a set of areas, each one associated with a different value) or *mailto*
- An *Alarm* can be implemented as a *text* (a text with font and style information) or an *audio_file*
- A *Description* can be implemented as a *text, image, audio, video, table*
- A *MultipleChoice* can be implemented as a *check_box* or a *list_box*
- A *Navigator* can be implemented as an *image_link, text_link, button*, *image_map*.
- A *NumericalEditFull* can be implemented as a *text_field* or a *spin_box* (a text field which includes also up and down buttons)
- A *NumericalEditInRange* can be implemented as a *text_field*, a *spin_box* or a *track_bar*
- A *PositionEdit* can be implemented as an *image_map*
- A *SingleChoice* can be implemented as a *radio_button*, *list_box*, *drop_down_list* or *image_map*
- A *TextEdit* can be implemented as a *text_field* or a *text_area*.



Each concrete refinement of the interactors has its own attributes for specifying the layout (borders, margin, padding, colour etc), together with an element which specifies the label, which can be either text, image or a combination of both (*button, check box, choice element, drop down list, image map, list box, mailto, radio button, spin box, track bar, table, text area, grouping, relation, repeater*).

The input modalities for this CUI are specified through the definition of two subclasses of *Event*: the *MouseEvent* and the *KeyboardEvent*, which model the occurrence of a mouse or keyboard input, respectively, abstracting from the window toolkit. The *MouseEvent* contains the representation of the typical information for the mouse pointer tracking: the position (*x* and *y* coordinates), the button pressed, and the wheel scroll size, while the *KeyboardEvent* contains the key code (which identifies the key pressed) and the *key_modifier* (which represents the *CTRL*, *SHIFT* and *ALT* buttons).

Having defined such information enables the designer to define the reaction of the concrete interactors at the user input. Each concrete implementation of the interactors contains in fact a list of mouse events and one event for keyboard input (if it makes sense having them for the interactor considered).

The list of mouse-related event is the following (all subtypes of the *MouseEvent* class):

- *mouse_enter*
- *mouse_move*
- *mouse_leave*

- *mouse_click*
- *mouse_hover*
- *mouse_double_click*

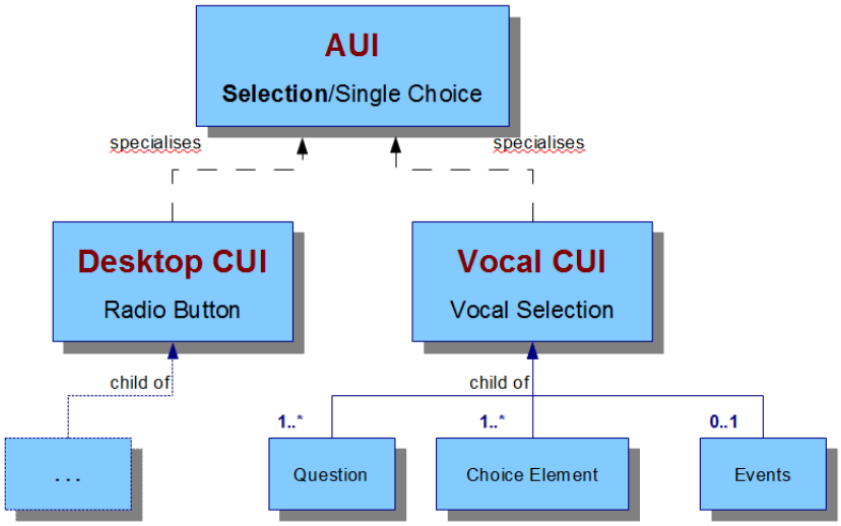
The keyboard-related events are (all subtypes of the *KeyboardEvent* class):

- key_down
- key_up
- key_press

5. Vocal Concrete User Interface

While in graphical interfaces the concept of presentation can be easily defined as a set of user interface elements perceivable at a given time (e.g. a page in the Web context), in the case of vocal interfaces we consider a presentation as a set of communications between the vocal device and the user that can be considered as a logical unit, e.g. a dialogue supporting the collection of information regarding a user. In defining the vocal language we have refined the abstract vocabulary for this platform. This mainly means that we have defined vocal refinements for the elements specified in the abstract language: interactors (user interface elements), the associated events and their compositions.

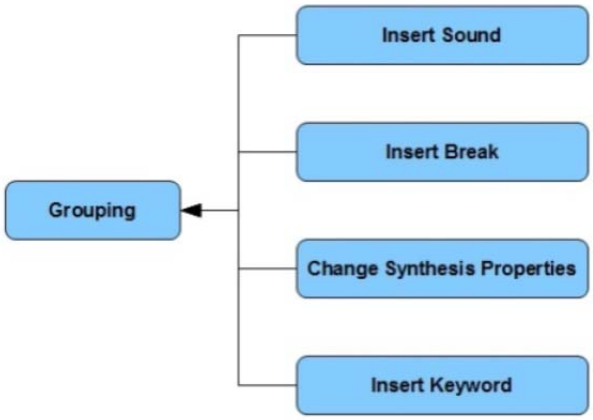
The refinement involves defining some elements that enable setting some presentation properties. In particular, we can define the default properties of the synthesized voice (e.g. volume, tone), the speech recognizer (e.g. sensitivity, accuracy level) and the DTMF (Dual-Tone Multi-Frequency) recognizer (e.g. terminating DTMF char). Only-output interactors simply provide output to the user; the abstract interface classifies them into text, description, feedback and alarm output. Refinement of the text element is composed of two new elements: *speech* and *pre-recorded message*. Speech defines text that the vocal platform must synthesize or the path where the platform can find the text resources. It is furthermore possible to set a number of voice properties, such as emphasis, pitch, rate, and volume as well as age and gender of the synthesized voice. Moreover, we have introduced control of behaviour in the event of unexpected user input: by suitably setting the element named *barge in*, we can decide if the user can stop the synthesis or if the application should ignore the event and continue. As mentioned above, the other element that refines abstract text is pre-recorded message: it defines the path of pre-defined audio resources that must be played. We support the case of missing resources by defining an *alternative content* that can be synthesized when this case occurs. Besides speech and pre-recorded message, the other only-output element is *sound.* This element permits defining the path of a non-vocal audio resource that must be played by the platform. It is also possible to insert a textual description of the sound that could be used as additional information regarding the sound content.



Selection interactors permit performing a selection between a set of elements; the abstract interface distinguishes between *single* and *multiple choice*. In order to support such interactions in vocal context we have introduced the interactor *vocal selection*. This element defines the question(s) to direct to the user and the set of possible user input that the platform can accept. In particular, it is possible to define textual input (word or sentences) or DTMF input. Depending on the type of selection one or multiple elements can be selected. Semantic differences between these two interactors are made at abstract user interfaces level: for the single choice it is possible to set only one selected element instead, in the case of the multiple choice, more than one user input can be set as selected elements. The control interactor at the abstract level can be distinguished in activator, to activate a functionality, and navigator, to manage navigation between the presentations. The activator is refined in the vocal language into: *command*, in order to execute a script, *submits*, to send a set of data to a server, and *goto* to perform a call to a script that triggers an immediate redirection. While the navigator is refined into: *goto*, for automatic change of presentation; *link*, for user-triggered change of presentation, and *menu* for supporting the possibility of multiple target presentations.

Edit interactors gather more complex user input. We refined three abstract elements: text edit, numerical edit and object edit. Text edit, from the graphical web context point of view, can be regarded as an editable textual field. In the vocal context we refined this concept with the *vocal textual input* element, which permits setting a vocal request and specifying the path of an external grammar for the platform recognition of the user input. Numerical edit is an interactor to collect numbers, which are refined into *numerical* input. As in the textual input it is possible to define a request and an external grammar. Moreover, it is also possible to set a predefined grammar, such as date, digits, phone or currency. Finally, we have refined the object edit interactor into a *record* element, which allows specifying a request and storing the user input as an audio resources. It is possible to define a number of attributes relative to the recording, such as *beep* to emit a sound just before recording, *maxtime* to set the maximum duration of the recording, and *finalsilence*, to set the interval of silence that indicates the end of vocal input. Record elements can be used for example when the user input cannot be recognised by a grammar (e.g. a sound). In the logical language one of the elements that permit the composition of the interactors is *grouping*. From the visual point of view we could refine this concept, for example, into a table element. Group vocal interactor is more complex. We propose four solutions to permit the user to identify the beginning and the end of a

grouping (see Figure below). Inserting a sound at the beginning and at the end for this purpose can be a good non-invasive solution. Another solution can be inserting a pause, which must be neither too short (useless) nor too long (slow system feedback). Moreover, it is possible to change the synthesis properties such as volume and voice gender. The last possibility is to insert keywords that explicitly define the start and the end of the grouping.



Another substantial difference of vocal interfaces is in the event model. While in the case of graphical interfaces the events are related mainly to mouse and keyboard activities, in vocal interfaces we have to consider different types of events: *noinput* (the user has to enter a vocal input but nothing is provided within a defined amount of time), *nomatch*, the input provided does not match any possible acceptable input, and *help*, when the user asks for support (in any platform specific way) in order to continue the session. All of them have two attributes: message, indicating what message should be rendered when the event occurs, and re-prompt, to indicate whether or not to synthesize the last communication again. It allows editing the various presentations in the central area. The user interface elements are created by drag-and-drop from the lists in the right frame, which show the elements that can be inserted according to the language specification. In the middle tab in the right frame it is also possible to specify the associated events and attributes. In the left side there is an interactive nested tree view of the presentations and the associated elements. The output of the tool is a logical description of the interface formalized in XML.

References

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.

F. Paternò, C. Sisti: Deriving Vocal Interfaces from Logical Descriptions in Multi-device Authoring Environments. Proceedings ICWE 2010, LNCS 6189, Springer Verlag, pp.204-217, Vienna, July 2010