OPEN Project

STREP Project FP7-ICT-2007-1 N.216552

Title of Document:	Logical User Interface Descriptions
Editor(s):	Carmen Santoro, Fabio Paternò, Giuseppe Ghiani
Affiliation(s):	CNR-ISTI
Contributor(s):	Susan-Marie Thomas
Affiliation(s):	SAP
Date of Document:	May 20, 2009
OPEN Document:	D2.3
Distribution:	EU
Keyword List:	User Interface Descriptions, Abstraction Levels
Version:	3.0

OPEN Partners:

CNR-ISTI (Italy)
Aalborg University (Denmrk)
Arcadia Design (Italy)
NEC (United Kingdom)
SAP AG (Germany)
Vodafone Omnitel NV (Italy)
Clausthal University (Germany)

Title: Logical User Interface Descriptions	**Id Number: D2.3**

Abstract

The purpose of this deliverable is to introduce and discuss relevant aspects of logical user interface descriptions, and explain how they can be useful in supporting the goals of the OPEN Project, in particular in the dynamic user interface adaptation and generation phases. Usually, such logical languages represent user interface models in XML-based format, which can also be obtained by automatic analysis of existing Web pages. In addition, they can be useful to obtain the design of versions adapted to various target platforms, which can be generated at run-time.

Table of Contents

1 Introduction

Logical user interface descriptions are specifications of user interfaces, which are able to abstract out some details and focus on the semantic aspects of a user interface, thus avoiding analysing a plethora of low level details, especially during the design phase. In fact, using multiple levels of abstractions is useful to identify and describe aspects that are relevant at the specific abstract level considered, which is particularly useful when designing interactive applications in multi-platform environments, as it happens with migratory interactive applications. The aim of this document is to describe the relevant aspects that logical user interface descriptions and related languages should address, how they can be useful and used in the OPEN Project, and the work carried out in this respect.

In order to do this, we found it useful to also include and report some work that has been done by ISTI-CNR under the aegis of SERVFACE, another EU project (http://www.servface.eu/). Indeed, within the latter project, a new language (called MARIA XML) for describing UIs at various levels of abstractions is currently being developed. Within OPEN project we plan to exploit the results of this language development within the scope of interactive migratory applications, and also provide feedback to ServFace regarding small improvements in the MARIA language that can be useful to address a wider set of user interface.

More in detail, in Section 2 we explain the motivations for adopting logical descriptions and related languages, while in Section 3 we provide a discussion of the state of the art in the field. In Section 4 we describe how logical user interfaces can be used to support migratory user interfaces. Then, while in Section 5 we illustrate the main features of the MARIA XML language, the rest of the deliverable is still dedicated to describe the work developed within the OPEN Project regarding logical User Interface descriptions. In Section 6 we show the requirements that have been raised by a case study in the game domain (Pacman) considered in OPEN, also providing in subsection 6.1 some concrete examples. Section 7 describes a number of aspects that we should address in order to meet the requirements raised by the business application under consideration in OPEN. In this context we are also considering KML, an XML-based language focused on geographic visualization. Finally, Section 8 is dedicated to summary and final conclusions.

2 Why Logical User Interface Descriptions

In the research community in model-based design of user interfaces there is a general consensus on what useful logical descriptions are (Calvary, Coutaz, Thevenin et al. 2003; Paternò 1999; Szekely 1996):

- *The task and object level*, which reflects the user view of the interactive system in terms of logical activities and objects manipulated to accomplish them;
- The *abstract user interface*, which provides a modality -independent description of the user interface;
- The *concrete user interface,* which provides a modality -dependent but implementation language independent description of the user interface;
- The *final implementation*, in an implementation language for user interfaces.

In order to better clarify which kind of information is associated with each of the above levels, we can have, for example, the *task* "select vegetarian menu" at the task and object level. This implies the need for a selection object at the abstract level, which indicates nothing regarding the platform and modality in which the selection will be performed (it could be through a gesture or a vocal command or a graphical interaction). When moving to the concrete description, a specific platform has to be assumed, for example the graphical PDA, and a specific modality-dependent interaction technique needs to be indicated to support the interaction in question (for example, selection could be through a radio-button or a drop-down menu), but nothing is indicated in terms of a specific implementation language. After choosing an implementation language, the last transformation from the concrete description into the syntax of a specific user interface implementation language is performed. The advantage of this type of approach is that it allows designers to focus on logical aspects and take into account the user's view right from the earliest stages of the design process.

The usefulness of using *logical* user interface descriptions is in the fact that they allow to focus on *semantic* aspects of user interfaces, removing the burden of analysing a plethora of low level details. In addition, the identification of multiple levels of abstractions is useful to identify (and group together) aspects that are relevant at the particular abstract level considered. The usefulness of exploiting multiple abstraction levels is particularly evident when designing interactive applications in multi-platform environments. In such environments, the advantage of having approaches based on such multiple levels of abstractions is that designers of multi-device interfaces do not have to learn all the details of the many possible implementation languages, because, through the logical descriptions, they can have full control over the design and leave the implementation to an automatic transformation from the concrete level to the target implementation language. Furthermore, if a new implementation language needs to be addressed, only the transformation from the associated concrete level to the new language has to be added, since all the concrete interface languages share the same structure. This is not difficult because the concrete level is already a detailed description of how the interface should be structured.

In addition, the UI descriptions at each abstraction level can be expressed in e.g. XML-based languages in order to make them more easily manageable and allow for their export/import in different tools. Another advantage of this approach is that if links are maintained among the elements in the various abstraction levels, this allows for the possibility of linking semantic information (such as the activity that users intend to do) and implementation levels, which can be exploited in many ways.

2.1 Using UI Descriptions at Design time and Runtime

The information contained in different XML-based descriptions can be used both at runtime and design time in the process of generating user interfaces.

At ***design time,*** the idea is to start with a single abstract vocabulary for all the different versions of a UI for the various platforms (e.g. desktop, mobile, vocal) the designers want to address. Then, by using refinement mechanisms, it will be possible to transform high-level abstraction descriptions to more concrete specifications, up to the final implementation. Such transformations can be carried out by taking into account information provided in some relevant models (user, platform, environment, etc). Data contained in such models can be more or less relevant, depending on the transformation (between two different abstraction levels) considered. For instance, depending on the characteristics specified in the model of the platform currently considered, an appropriate transformation will be supported for obtaining a concrete user interface from an abstract one, so as to take into account the features and resources of the device at hand. Moreover, it is worth pointing out that the transformations between the different abstraction levels should also be available for performing the inverse process (e.g. from concrete specifications to abstract ones), in order to maintain the models consistent.

The information contained in such specifications/models are also useful ***at runtime***, since they are considered when a dynamic adaptation is needed as a consequence of some occurring events. For example, if the user changes a device, the adaptation phase has to calculate again the effects of the changes occurred, and then trigger a dynamic generation of the user interface so as to adapt the UI to the new device. In this case, a possibility of the adaptation phase is to carry out a semantic redesign of the UI: the current implementation language used in the current device can be used to access the related concrete description which has to be transformed in order to obtain a concrete UI description for the new device, from which the new final implementation for the new device can be obtained.

3 State of art and Evolution of the Field

Some languages have been put forward for expressing the aspects relevant when describing user interfaces. The eXtensible Interface Markup Language (XIML) (http://www.ximl.org/) is a XML-based language for specifying multiple models in a model-based approach. Developed by a forum headed by RedWhale Software, it was introduced as a solution enabling universal support of functionality across the entire lifecycle of a user interface. The structure of XIML consists of components, relations, and attributes. In XIML 1.0 there are five basic interface *components*: i)*task* (user tasks that the interface supports), ii)*domain* (organized collection of data objects and classes of objects viewed/manipulated by a user), iii)*user* (a user group or an individual user.), iv)*dialog* (the interaction actions available to the user), and v)*presentation* (it defines a hierarchy of interaction elements including concrete UI objects like e.g. windows, buttons, sliders,..). One of the main disadvantages of XIML is that a rather simple notion of task models is supported by this approach, for which tool support is not currently available. Another drawback is the fact that it is developed within a software company, so its use is protected by copyright.

User Interface Markup Language (UIML) (http://www.uiml.org/ ; Abrams et al., 1999) was one of the first model-based languages targeting multi-device interfaces. Developed by Harmonia, it structures the user interface in: i)*structure* (which contains a list of <part> elements describing some abstract portions of a UI), ii)*style* (it contains a list of <property> elements, giving presentation properties to the parts), iii)*content* (it contains text, images, and other content that goes into the UI), iv)*behaviour* (it contains a set of rules to define how the UI reacts to external stimula, either from a user or from the external programming environment), v)(*logic* (it defines the application programming interface (API) of business logic with which the UI interacts) and vi)*presentation* (generally, it simply names a vocabulary file) sections. One of the main disadvantages of UIML is the fact that it does not support the task level. Also, it has not been applied to obtain rich multimodal UIs.
Another approach is Personal Universal Controller (PUC) environment, which supports the downloading of logical descriptions of appliances and the automatic generation of the corresponding user interfaces (Nichols et al. 2002). A PUC first downloads a specification of the appliance’s functions, and then it automatically creates an interface for controlling them. To connect a PUC to any real appliance, an appliance *adaptor* should be built for each proprietary appliance protocol the PUC communicates with. A PUC engages in two-way communication with everyday appliances: a PUC can send messages requesting the value of a state, or the invocation of a command, while the appliance can send the specification or the current value of a particular state. The PUC architecture has been designed to be independent of the type of interface presented to the user. *Generators* have been developed for two different types of interfaces: graphical and speech. One of the main disadvantages of the PUC approach is that its application area is limited to home appliances that require similar interfaces.

Another language that is based on multiple-layer structure is UsiXML. The semantics of the UsiXML models (Limbourg, Vanderdonckt, Michotte et al. 2004; Vanderdonckt, 2005) are based on meta-models expressed through UML class diagrams, from which the XML Schema definition are systematically derived. The authors use graph transformations for supporting model transformations, which is an interesting academic approach with some performance issues. There are some tools supporting UsiXML, e.g. a translator from UsiXML specification to Flash, a tool allowing for sketching UIs, a tool for performing task analysis. Differently from XIML, UsiXML is freely available and not protected by copyright.
XForms (http://www.w3.org/MarkUp/Forms/) represents a concrete example of how the research in model-based approaches has been incorporated into an industrial standard. XForms is an XML language for expressing the next generation of Web forms, through the use of abstractions to address new heterogeneous environments. The language includes both abstract and concrete descriptions, and it supports the definition of a data layer inside the form. User interface controls encapsulate relevant metadata such as labels, thereby enhancing accessibility of the application when using different modalities. The list of XForms controls includes objects such as *select* (choice of one or more items from a list), *trigger* (activating a defined process), *output* (display-only of form data), *secret* (entry of sensitive information), etc. Through the use of the appearance attribute XForms refers to concrete examples like radio buttons, checkboxes, etc. XForms is mainly used for expressing form-based UIs and less for supporting other interaction modalities, such as voice interaction. Another language for describing user interfaces is TERESA XML (Mori, Paternò and Santoro, 2004). Developed at ISTI-CNR, together with the associated TERESA tool, under the aegis of an European FP5 Project (CAMELEON), it supports the various possible abstraction levels for supporting the design and development of multidevice applications. The *task level* describing the activities that should be supported by the system, is specified using the CTT notation (Paternò 1999; Berti et al., 2004). At the abstract level, a UI is composed of a number of presentations and connections among such presentations and mainly defining the UI dynamic behaviour. *Abstract* interactors indicate the possible interactions in a platform-independent manner (eg., selection, editing, etc.). This abstract level also describes how to compose such basic elements through *composition operators*, which are: i)*Grouping*: indicates a set of interface elements logically connected; ii) *Relation*: highlights a one-to-many relation among some elements, one element has some effects on a set of elements; iii) *Ordering*: it expresses an ordering among a set of elements; iv) *Hierarchy*: different levels of importance can be defined among elements. The *concrete* level is a refinement of the abstract user interface: depending on the type of platform considered, there are different ways to render the various interactors and composition operators of the abstract user interface. For example, a navigator (an abstract interactor) can be implemented either through a textlink, or an imagelink, or a simple button, and in the same way, a single choice object can be implemented using either a radio button or a list box or a drop-down menu. The same holds for the operators: for instance, the

grouping operator in a desktop platform can be refined at the concrete level by unordered lists by row/column, but also through fieldsets, bullets, and colors. In the final level, a specific implementation language (e.g. XHTML, Java, etc.) has to be used for the considered platform. Some evaluation experiences (Chesta et al., 2004) highlighted limitations on this language, which pushed the development of an evolution of it (called MARIA XML), which is currently under development within another European Project (SERVFACE, http://www.servface.org). The main novel features of MARIA XML will be described in Section 5, while some examples of using TERESA XML language (which is progressively evolving to MARIA XML) for specifying interactive migratory applications will be described in Section 6.1.

All these approaches have stimulated a good deal of interest. An indication is a number of initiatives that have started to define international standards in the area (e.g.: New W3C Group on Model-based User Interfaces: http://www.w3.org/2005/Incubator/model-based-ui/) or to define their adoption in industrial settings (e.g.: Working Group in NESSI NEXOF-RA IP).

4 How logical UI descriptions can be exploited for Migratory User Interface

As mentioned in previous sections, logical UI descriptions are useful in that they allow UI designers to focus on relevant aspects of the UI and therefore they enable them to take design decisions without being lost in too many implementation details. In addition, logical descriptions also support interoperability between different implementation languages.

Figure 1 shows how the abstraction layers that are exploited to support migratory user interfaces. The current architecture assumes that a desktop Web version of the application front-end exists (a reasonable assumption given the wide availability of this type of application). Then, starting from the UI implemented for the source device (see “Pre-computed Implement.1” box in Figure 1) the migration server automatically generates a version adapted to the target device (namely, it *redesigns* the UI for the source device into another UI for the target device). For this purpose, it first creates the logical Concrete UI (CUI) descriptions of the source version through a reverse engineering process, which is applied to the page currently accessed by the user. Then, the server performs a semantic redesign [Paternò et al., 2008] of such a CUI for creating a logical concrete description adapted to the target device. Thus, such *semantic redesign* transformations involves i) preserving the semantics of the user interactions after the change of device so that the UI allows the users to continue to carry out their activities and ii) obtaining a Concrete User Interface adapted to the characteristics/resources of the target device. The second aspect is not trivial because it may happen that some task is not supported by the target device (e.g. a long video cannot be rendered with a limited mobile phone). For all the tasks that can be supported the semantic redesign identifies concrete techniques that preserve the semantics of the interaction but supports it with techniques most suitable for the new device (for example in mobile devices it will replace interactors with others that provide the same type of input but occupying less screen space). In the case of desktop-to-mobile transformation a page splitting algorithm based on the logical structure of the considered page and the cost of the user interface elements is also applied.

Therefore, the semantic redesign process is a transformation process that translates a Concrete User Interface for a certain (source) platform into a correspondent Concrete User Interface for a certain (target) platform, while supporting the semantic of the interaction. Of course, the last step for obtaining a final implementation for the target device (see “Generated Implement.2” box in Figure 1) is translating the CUI into the specific constructs of the implementation language (e.g. Java, XHTML, ..) considered in the target device.

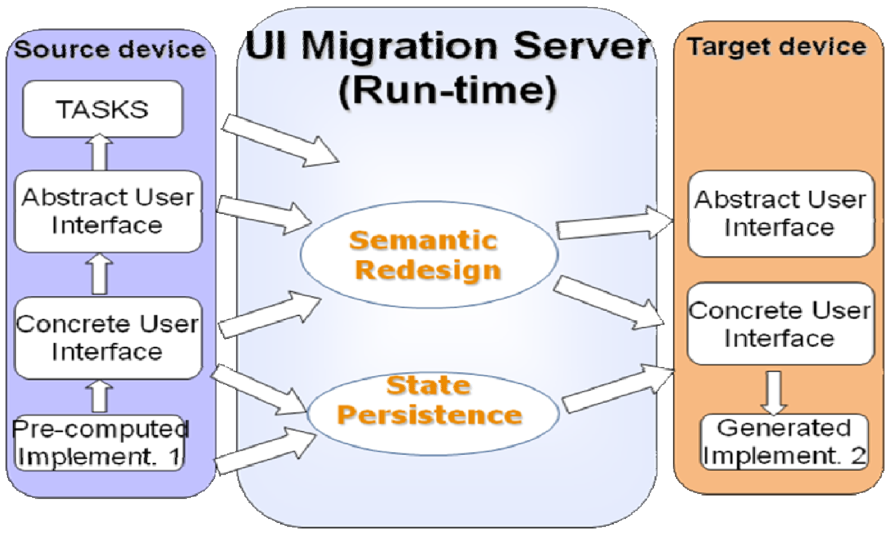


Figure 1: The Different Logical UI Levels used for supporting Migratory Interfaces

5 MARIA XML (Contribution from the SERVFACE Project)

This section mainly presents work that has not been developed within the OPEN project. Indeed, MARIA XML (Model-based Authoring environment for Interactive Applications) is a new UI specification language that is currently developed at the HIIS laboratory of ISTI-CNR as a part of the SERVFACE EU STREP Project (http://www.servface.org). Within OPEN project we mainly plan to *apply* this language to dynamically specify migratory applications and, if necessary, request small changes to the language to better apply it to a widers et of user interfaces.

The goal of MARIA XML is providing the capabilities to model new forms of human-computer interaction. It can be used for the abstract and concrete user interface definition. As we already briefly mentioned before, MARIA XML is an evolution of another language (TERESA XML), also developed at ISTI-CNR. As an example of requirement which was not addressed in TERESA XML is the lack of flexible support for UI events handling. This limitation together with several other features judged necessary for addressing the improved interactivity and flexibility of more recent UIs, led to the decision of adopting of a new language (MARIA XML) able to overcome such limitations. In addition, some exercise in modeling the case studies of the OPEN project provided useful feedback to the ServFace project in order to identify aspects that the new MARIA XML language should address.

MARIA XML inherits the modular approach of TERESA XML with one language for the abstract description and then a number of platform dependent languages that refine the abstract one depending on the interaction resources considered. In its first version we have considered the following platform: graphical form-based, graphical mobile form-based, vocal, digital TV, graphical direct manipulation, multimodal (graphical and vocal) for desktop and mobile, IPhone (with support for multi-touch and accelerometers). A number of features have been included in MARIA XML language. In the next subsections we will briefly describe them.

5.1 Main Added Features

5.1.1 Introduction of Data Model.

We have introduced an abstract description of the underlying data model of the user interface, which is necessary to represent the data (types, values, etc.) handled by the user interface. Indeed, by means of defining an Abstract Data Type/Data Model, the interactors (the elements of the abstract or concrete user interface) composing an abstract [concrete] user interface, can be bound to a specific type or an element of a type defined in the abstract [resp.:concrete] data model. The introduction of a data model also allows for more control on the admissible operations that will be done on the various interactors. In MARIA XML, the data model is described using the XSD type definition language. Therefore, the introduction of the data model can be useful for: correlation between the value of interface elements, conditional presentation connections, conditional layout of interface parts, specifying the format of the input values, application generation from the interface description.

5.1.2 Introduction of an Event Model

In addition, an event model has been introduced, at different abstract/concrete levels of abstractions. The introduction of an event model allows for specifying at different abstraction levels how the user interface is able to respond to events triggered by the user. In MARIA XML two types of events have been introduced:

- Property change events: events which change the status of some UI properties. The handlers for this type of events are only change_properties, which indicate in a declarative manner how and under what conditions to change property values
- Activation events are events raised by activators, which are interactors with the purpose to activate some application functionality (e.g. access to a database or to a web service). This type of event can have both change_properties or script handlers (which have associated a generic script). This kind of event can be associated only to activators interactors.

5.1.3 Supporting Ajax scripts, which allow continuously updating of fields

Another aspect that has been included in MARIA XML is the possibility of supporting continuously updating of fields at the abstract level. We have introduced an attribute to the interactors: continuosly-updated= "true"["false"]. The concrete level has the duty to provide more detail on this feature, depending on the technology used for the final UI (Ajax for web interfaces, callback for standalone application etc.). For instance, with Ajax asynchronous mechanisms, there is a behind-the-scene communication between the client and the server about what has to be modified in the presentation, without explicit request from the user. When it is necessary the client redraws the relevant part rather than redrawing the entire presentation from scratch: this allows for quicker changes and real-time updates.

5.1.4 Dynamic set of user interface elements

Another feature that has been included in MARIA XML is the possibility to express the need of dynamically changing only a part of the UI. This has been specified in such a way to be able to affect both how the UI elements are arranged in a single presentation, and how it is possible to navigate between the different presentations. The possibility to change only a part of a presentation has been introduced. Therefore, the content of a presentation can dynamically change (this is also useful for supporting ajax techniques). In addition, it is also possible to specify a dynamic behaviour that changes depending on specific conditions: this has been implemented thanks to the use of conditional connections between presentations. In the next sections we will provide a more detailed description of concepts and models that have been included in MARIA XML, both for the Abstract UI and the Concrete UI.

5.2 MARIA XML-AUI

Focusing on the abstract level, it is generally recognised that the main benefits in using an abstract description of a user interface are for designers of multi-device interfaces, because they do not have to learn all the details of the many possible implementation languages supported by the various devices. Instead, they can reason in abstract terms without being tied to a particular platform/modality/implementation language. In this way, they have the possibility to focus on the *semantic* of the interaction (what the intended goal of the interaction is), regardless of the details and specificities of the particular environment considered.

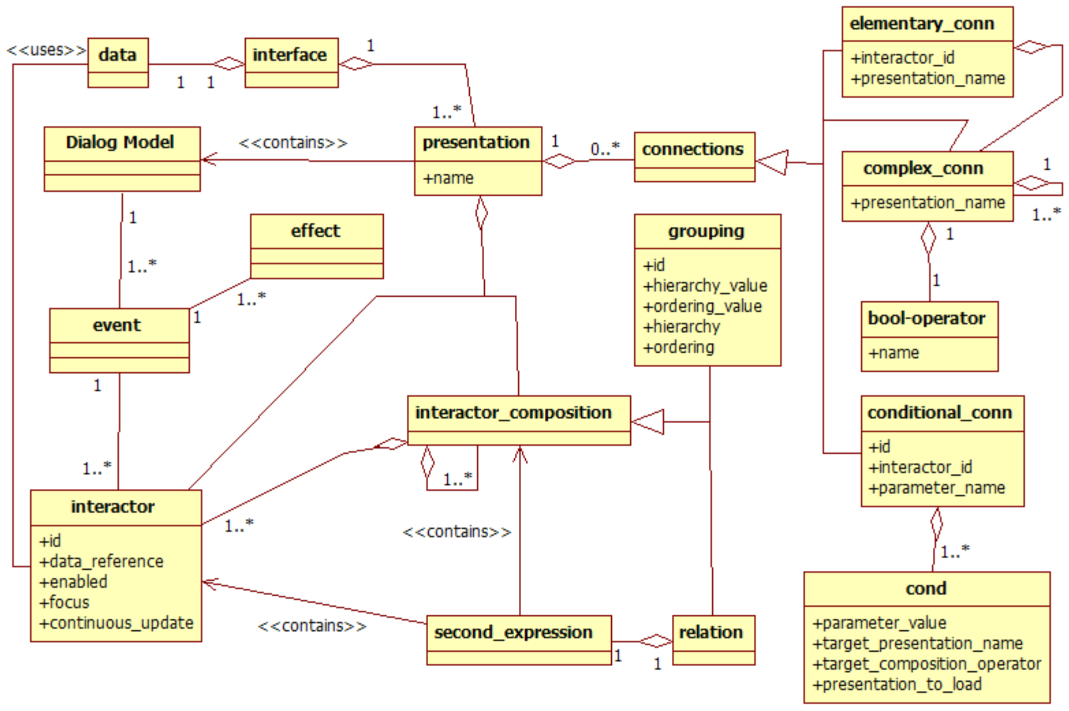


Figure 2: The Abstract UI in MARIA XML (only the higher levels of the hierarchy have been unfolded)

Figure 2 shows the main elements of the abstract user interface meta-model (some details have not been shown for readability reasons) of MARIA XML. As you can see, an interface is composed of one data model and one or more presentations. Each presentation is composed of name, a number of possible connections and an interactor expression which can be either an elementary interactor or an interactor composition. The presentation contains also a dialog model, which contains information about the events that can be triggered by the presentation in a given time. The dynamic behaviour of the events is specified using the CTT temporal operators (for example, the event1 and the event2 can be executed in parallel, or in mutual exclusive choice, or sequentially, etc.). When an event occurs, it produces a set of effects (such as performing operations, calling services etc.) and can change the set of currently enabled events (e.g. an event occurring on an interactor can affect the behavior of another interactor, by e.g. disabling the availability of an event associated to another interactor). The dialog model can also be used to describe parallel interaction between the user and the interface.

A *connection* indicates what the next active presentation will be when a given interaction is performed and it can be either an elementary connection, or a complex connection (when a Boolean operator composes several connections) or a conditional connection (when various conditions on connections are specified).

There are two types of interactor composition: *grouping* or *relation*, the latter has at least two elements (interactor or interactor compositions) that are in relation each other. An *interactor* (see Figure 3) can be either an interactor object or an only_output object. The first one can assume one of the following types: *selection*, *edit*, *control*, *interactive description*, depending on the type of activity the user is supposed to carry out through such objects. An *only_output* interactor can be *object*, *description*, *feedback*, *alarm*, *text*, depending on the supposed information that the application provides to the user through this interactor.

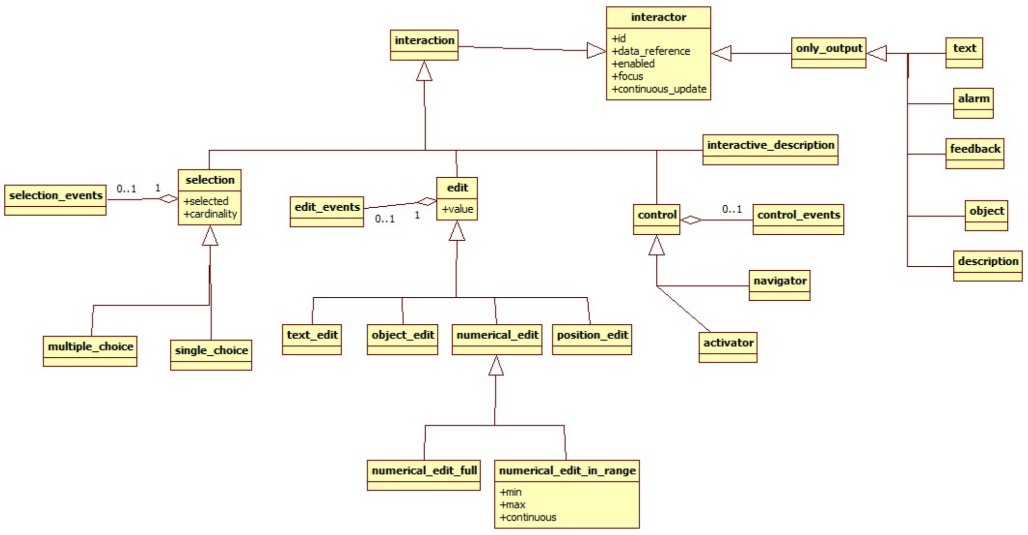


Figure 3: Unfolding the "interactor" element in MARIA XML (AUI)

The selection object is refined into *single_choice* and *multiple_choice* depending on the number of selections the user can perform. It is worth pointing out that the further refinement of each of these objects can be done only by specifying some platform-dependent characteristics, therefore at the concrete level (see next section for an example). The edit object can be further refined at the abstract level into text_edit, object_edit, numerical_edit and position_edit depending on the type of effect desired. A more refined indication of the elements that can be edited is obtained through the use of the data model. The control object is refined into two different interactors depending on the type of activity supported (navigator:

navigate between different presentations, activator: trigger the activation of a functionality). It is worth pointing out that all the interaction objects have associated events in order to manage the possibility for the user interface to model how to react after the occurrence of some events in their UI. The events differ depending on the type of object they are associated with.

5.3 MARIA XML-CUI

In Figure 4 you will find how an interaction element of type "multiple_choice" (which is a leaf in the abstract UI model) is expanded at the concrete level in MARIA XML language.

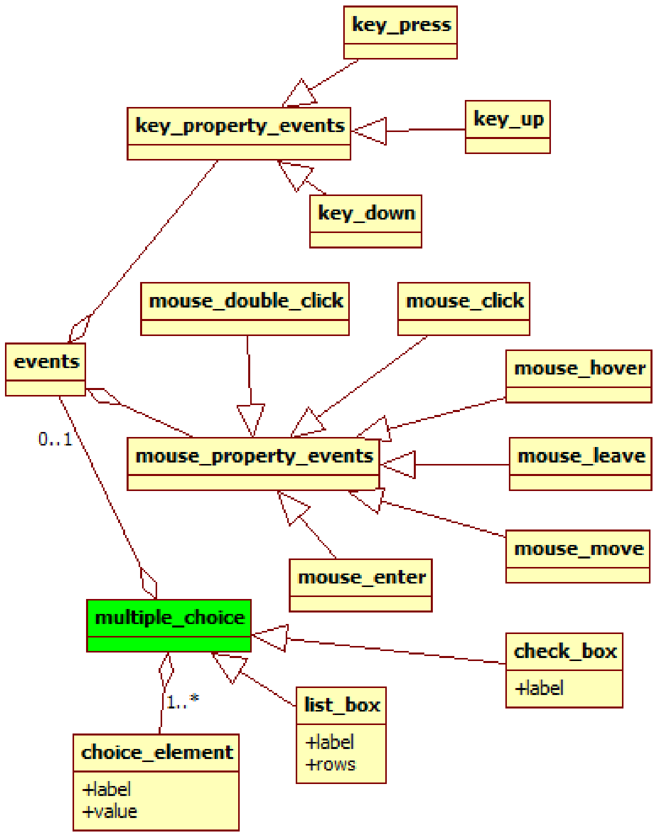


Figure 4: Unfolding the "multiple_choice" element in MARIA XML (CUI-desktop)

As you can see from Figure 4, the *multiple_choice* element has a number of choices (choice_element) and each of them has two attributes (label and value). Two possible elements can support a multiple selection (when considering a desktop platform): *list_box* and *check_box*. In addition, a number of events can be defined on this object and they can be of two different types: events that are generated by interacting with the keyboard (*key_property_events*) and events that can be generated by the mouse (*mouse_property_events*). Different elements can be derived from each of such group. For instance, key_press, key_up and key_down are all sub-types of the key_property_events type.

6 Requirements raised by the PacMan case study in OPEN

In OPEN some requirements were raised regarding the logical user interfaces descriptions, in particular, when analysing a version of the Pacman game (further details in [OPEN D5.1]) considered as one of the case studies in the Project. Indeed, when trying to apply the features of TERESA XML, some requirements were raised because there were some features that was not possible to support with the old version of such a language.

For instance, a table was not included as an additional grouping technique (only data table appeared in TERESA XML language). Instead, a table can also be used for grouping together different elements, and this is the case when it is basically used as a *layout table*. Therefore, the possibility to use a table as an additional grouping technique has to be considered. In addition, there also was the need of having nested tables (which is another aspect that occurs in the Pacman example) and even with empty elements: this has been added in the new version of the language. In addition, in the Pacman example there was an imagemap element which has to support on the one hand the selection (performed by the user) of the pacman direction and, on the other hand, the activation of a function (belonging to the application logic part) which has to react appropriately to the newly selected direction of the Pacman. This element was not supported in TERESA XML and it was another requirement posed to the language and highlighted by the Pacman experiment. In addition, within the Pacman there is the possibility to select (via a radiobutton) a specific type of maze, or a specific animation speed for the Pacman and, associated to the selection of each choice there is also the activation of a function (script). The handling of an event and its specification within the language also represented a further requirement for revising TERESA XML.

6.1 Example Specifications in this case study

In this section we show an example application of our approach, which supports a migratory Pacman game. This case study contributed to discover some UI aspects that were not supported in the version of the UI description language used at that time (as it has been described before, in Section 6). At the delivery date of this document we are still using an evolution of the TERESA XML language in the software support for migratory user interfaces, with the aim of progressively moving to MARIA XML, which is more powerful for supporting migratory user interfaces, thanks to the foreseen provision for e.g. data model, event model, and more interactive and complex applications (e.g. RIA applications).

Therefore, the UI language example specifications reported in this section reflect the current state of the TERESA XML language specifications.

As we mentioned, imagemap (XHTML <map> tag) was an example of element not supported in TERESA XML. Also, the possibility of handling events able to activate some functions in the application was not sufficiently supported. Indeed, in the desktop version, the Pacman can be controlled either using the keyboard (e.g. arrow keys can be used to specify directions) or by using an imagemap. In the latter case, the user can click on a specific region of the image or even pass the mouse over such a region (event “onMouseOver”) in order to activate a function allowing to set the new direction that the Pacman should take. Below (Listing 1) there is the excerpt from the XHTML Pacman page, describing the imagemap element which allows the user to control the game:

```
<img src="images/ctrl.gif" border="0" height="72" width="72"
usemap="#ctrlmap" vspace="1" alt="m"/>
 <map name="ctrlmap" id="ctrlmapid">
  <area shape="rect" coords="28,27,44,44"
   href="javascript:doPause()"
   onmouseover="window.status=' ';return true"
   onmouseout="window.status=' ';return true" alt="m"/>
<area shape="poly"
coords="71,17,71,17,71,54,62,62,44,44,43,29,62,9"
href="javascript:move(1)"
onmouseover="move(1);window.status='right.'; return true"
onmouseout="window.status=' '; return true" alt="m"/>
                <area shape="poly"
coords="0,19,0,19,11,9,27,28,28,44,11,62,2,53"
href="javascript:move(2)"
onmouseover="move(2);window.status='left.'; return true"
onmouseout="window.status=' '; return true" alt="m"/>
                <area shape="poly"
coords="16,71,16,71,1,52,26,39,45,39,71,54,53,71"
href="javascript:move(8)"
onmouseover="move(8);window.status='down.'; return true"
onmouseout="window.status=' '; return true" alt="m"/>
                <area shape="poly"
coords="2,16,2,16,18,0,54,0,71,17,45,38,27,38"
href="javascript:move(4)"
onmouseover="move(4);window.status='up.'; return true"
onmouseout="window.status=' '; return true" alt="m"/></map>
```

Listing 1: An excerpt from the XHTML Pacman page (XHTML imagemap)

In Listing 2 you can see how the corresponding imagemap element can be expressed at the concrete level:

```
<imagemap usemap="#ctrlmap">
  <image alt="m" height="72" src="images/ctrl.gif" width="72" />
   <map name="ctrlmap">
     <area alt="m" coords="28,27,44,44" shape="rect"
      target="javascript:doPause()">
      <trigger>
              <event event_name="onmouseout" />
              <effect name="window.status=' '; return true" />
      </trigger>
      <trigger>
              <event event_name="onmouseover" />
              <effect name="window.status=' '; return true"/>
```

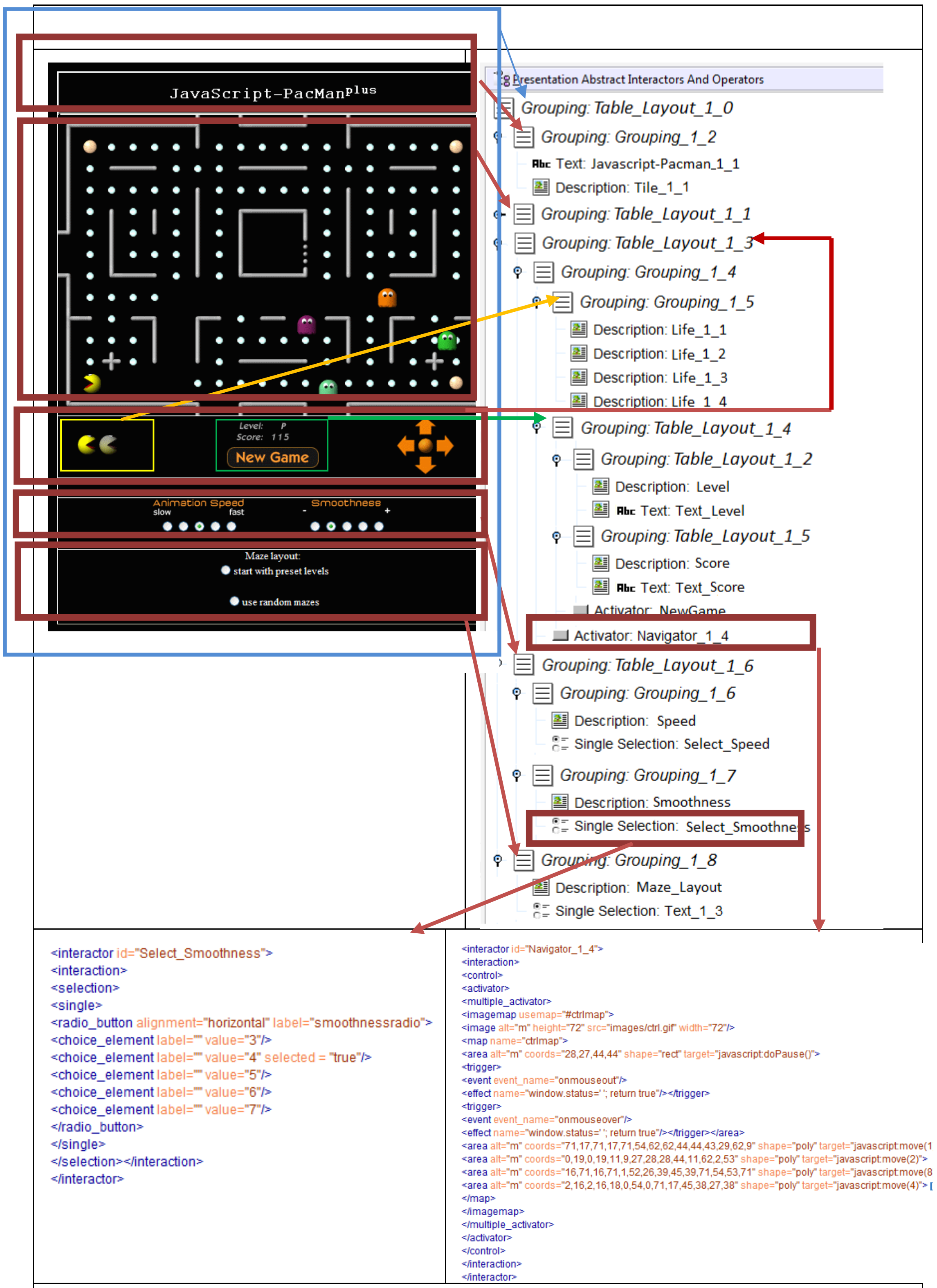
```
          </trigger></area>
        <area alt="m" coords="71,17,71,17,71,54,62,62,44,44,43,29,62,9"
shape="poly" target="javascript:move(1)">
          <trigger>
                    <event event_name="onmouseout" />
                    <effect name="window.status=' '; return true" />
          </trigger>
          <trigger>
                    <event event_name="onmouseover" />
                    <effect name="move(1);window.status='right.'; return true"/>
          </trigger> </area>
          <area alt="m" coords="0,19,0,19,11,9,27,28,28,44,11,62,2,53"
shape="poly" target="javascript:move(2)">
          <trigger>
                    <event event_name="onmouseout" />
                    <effect name="window.status=' '; return true" />
          </trigger>
          <trigger>
                    <event event_name="onmouseover" />
                    <effect name="move(2);window.status='left.'; return true"/>
          </trigger> </area>
        <area alt="m" coords="16,71,16,71,1,52,26,39,45,39,71,54,53,71"
shape="poly" target="javascript:move(8)">
          <trigger>
                    <event event_name="onmouseout" />
                    <effect name="window.status=' '; return true" />
          </trigger>
          <trigger>
                    <event event_name="onmouseover" />
                    <effect name="move(8);window.status='down.'; return true" />
          </trigger></area>
          <area alt="m" coords="2,16,2,16,18,0,54,0,71,17,45,38,27,38"
shape="poly" target="javascript:move(4)">
            <trigger>
                    <event event_name="onmouseout" />
                    <effect name="window.status=' '; return true" />
            </trigger>
            <trigger>
                    <event event_name="onmouseover" />

                    <effect name="move(4);window.status='up.'; return true" />

            </trigger></area>

          </map></imagemap>
```

Listing 2: The specification of the imagemap element at the CUI level



```
Presentation Abstract Interactors And Operators
Grouping: Table_Layout_1_0
  Grouping: Grouping_1_2
    Text: Javascript-Pacman_1_1
    Description: Tile_1_1
  Grouping: Table_Layout_1_1
  Grouping: Table_Layout_1_3
    Grouping: Grouping_1_4
      Grouping: Grouping_1_5
        Description: Life_1_1
        Description: Life_1_2
        Description: Life_1_3
        Description: Life_1_4
      Grouping: Table_Layout_1_4
        Grouping: Table_Layout_1_2
          Description: Level
          Text: Text_Level
        Grouping: Table_Layout_1_5
          Description: Score
          Text: Text_Score
        Activator: NewGame
      Activator: Navigator_1_4
  Grouping: Table_Layout_1_6
    Grouping: Grouping_1_6
      Description: Speed
      Single Selection: Select_Speed
    Grouping: Grouping_1_7
      Description: Smoothness
      Single Selection: Select_Smoothness
  Grouping: Grouping_1_8
    Description: Maze_Layout
    Single Selection: Text_1_3
```

```xml
<interactor id="Select_Smoothness">
<interaction>
<selection>
<single>
<radio_button alignment="horizontal" label="smoothnessradio">
<choice_element label="" value="3"/>
<choice_element label="" value="4" selected = "true"/>
<choice_element label="" value="5"/>
<choice_element label="" value="6"/>
<choice_element label="" value="7"/>
</radio_button>
</single>
</selection></interaction>
</interactor>
```

```xml
<interactor id="Navigator_1_4">
<interaction>
<control>
<activator>
<multiple_activator>
<imagemap usemap="#ctrlmap">
<image alt="m" height="72" src="images/ctrl.gif" width="72"/>
<map name="ctrlmap">
<area alt="m" coords="28,27,44,44" shape="rect" target="javascript:doPause()">
<trigger>
<event event_name="onmouseout"/>
<effect name="window.status=' '; return true"/></trigger>
<trigger>
<event event_name="onmouseover"/>
<effect name="window.status=' '; return true"/></trigger></area>
<area alt="m" coords="71,17,71,17,71,54,62,62,44,44,43,29,62,9" shape="poly" target="javascript:move(1
<area alt="m" coords="0,19,0,19,11,9,27,28,28,44,11,62,2,53" shape="poly" target="javascript:move(2)">
<area alt="m" coords="16,71,16,71,1,52,26,39,45,39,71,54,53,71" shape="poly" target="javascript:move(8
<area alt="m" coords="2,16,2,16,18,0,54,0,71,17,45,38,27,38" shape="poly" target="javascript:move(4)"> [
</map>
</imagemap>
</multiple_activator>
</activator>
</control>
</interaction>
</interactor>
```

Figure 5: Mapping XHTML constructs of Pacman example to the CUI specification.

In particular, you can see how the events associated to each region of the imagemap have been modelled through the two elements “event” (the name of the event) and “effect” (which models the consequence that the event triggering will cause on the UI). Another feature that was included as a consequence of the analysis of the requirements of the Pacman was the possibility to specify nested tables. In Figure 5 you can see how it is possible to specify nested tables in TERESA XML: for instance, as you can see, the table Table_Layout_1_6 is a table having two cells (each cell contains a grouping of an image describing a radiobutton (e.g.: “Animation Speed” or “Smoothness” in the Pacman example) and the corresponding radiobutton with some options. However, such a table is in turn a row of the larger table Table_Layout_1_0, which contains the entire page, therefore there is a nesting of tables supported by the language. In addition, in Figure 5 the correspondences between the XHTML Pacman, the CUI structure and some excerpts of the CUI in the concrete language are visualised.

7 Requirements raised by the business case study

While Section 6 discussed some specific requirements raised by the PacMan case study, this section covers the research challenges raised by the business case study. In that sense it serves as a preview of some potential future research directions for the project.

The challenges raised by the business use case are divided into two types: those related to dimensions of complexity, and those related to the decision to implement the applications as Rich Internet Applications (RIAs). Each is discussed in a separate sub-section.

The dimensions of complexity that have been identified are all related to UI migration, as opposed to application logic migration, since so far the scenario requires only UI migration. The identified dimensions are:

- number of users of one device,
- whether a change of modality is required,
- number of applications involved,
- and how presentation and control elements of multiple applications are combined.

Each of these categories is discussed in the following sub-sections. But, first, the business scenario from D5.1 is generalized to widen its scope to include a broad range of business applications. This shows the importance of the scenario for businesses, and also raises new challenges that must be met to enable migration of such applications.

7.1 Generalization of the Business Scenario

The D5.1 business scenario describes a specific scenario from the field of emergency response management. Such management has become more important as a confluence of forces, including climate change, resource shortages, environmental pollution, inequality and social unrest, has resulted in an increase in catastrophes and emergency situations.

Increased attention to this subject is also reflected by coordination activities like the 4th Europäischer Katastrophenschutzkongress, 4th European Congress on Civil Protection (www.civil-protection.com). Another indicator of its importance is spending on research programs which include projects like SoKnos (www.soknos.de) which researches the use of IT to increase the effectiveness and coordination of organizations involved in emergency response. The OPEN business scenario is, in fact, inspired by scenarios from the SokNos project, of which SAP is a partner.

This emergency scenario can, however, also be generalized to more typical situations that come to mind when the word *business* is used, such as new product design, financial decision making, and all kinds of planning efforts from manufacturing and supply to sales and distribution. The rest of this section makes this generalization in order to provide a more general basis for drawing out the challenges which it raises for UI migration.

The business scenario has three major phases that can be generalized as follows. First, a number of experts work independently on the same problem from their respective points of view. Each uses his own device to arrive at his results.

Next they come together in a central planning center and share their results (through UI migration) on a wall-sized screen (a smart wall) which may be multi-touch, which means that multiple people can interact with it simultaneously. Figure 6 from D5.1, depicts two experts migrating their results to the EOC (Emergency Operations Center) smart wall. By sharing, e.g., overlaying results, they can interact to develop and refine plans that constitute a more effective response to the problem as a whole. Often the planning will involve geographic considerations, so maps will play a central role in the presentation of information and actions.

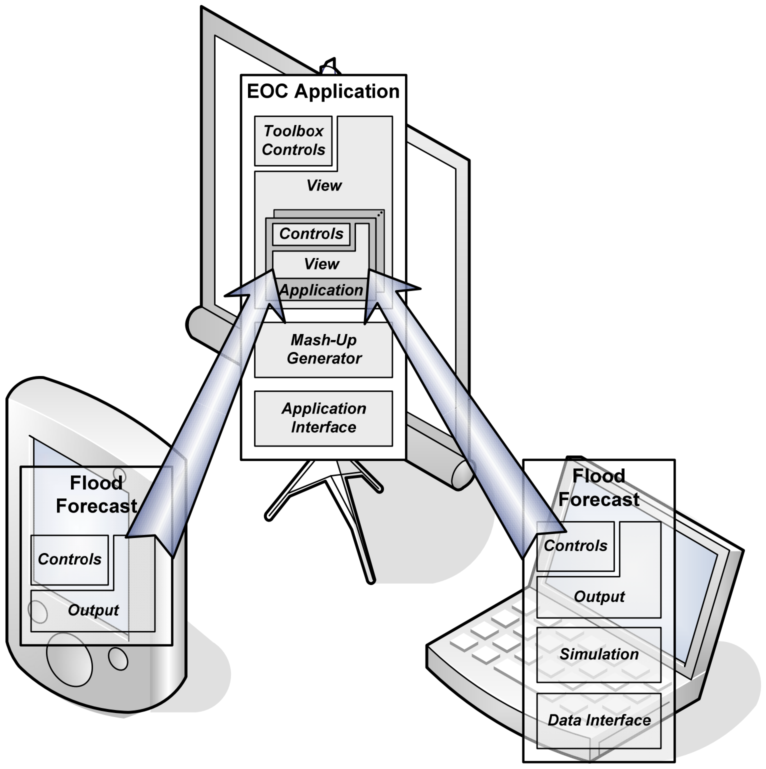


Figure 6: Migration of a two flood forecast simulations from a PDA and a laptop to a high-resolution, multi-touch smart wall.

During the second phase of the scenario the experts jointly refine their analyses and plan a response to the problem. In this phase the smart wall serves as a planning and communications center, bringing together all the communications concerning the problem.

An extension of the original scenario could be that all verbal and email communications from the field to the planning center are captured and maintained by means of specially configured devices used by the staff in the field. These could transmit a recording of the audio communication (or a copy of the email) together with GPS coordinates, the user's identity, the type of the communication and its priority to the planning center. In this way all communications are available for review and action and, given the GPS data, can be displayed on the planning map if desired.

The third phase of the scenario begins after an initial response has been planned. Its purpose is to alert responsible managers to changes in circumstances that may require a reconsideration of the plans.

Alerting is done via a personal device like a cell phone, so this phase involves duplication of some UI elements from the shared smart-wall to a personal device, as depicted in Figure 7. This phase may also involve a change of modality from visual to audio, so that the manager can listen to what is happening. This is essential if the manager is underway in a car.

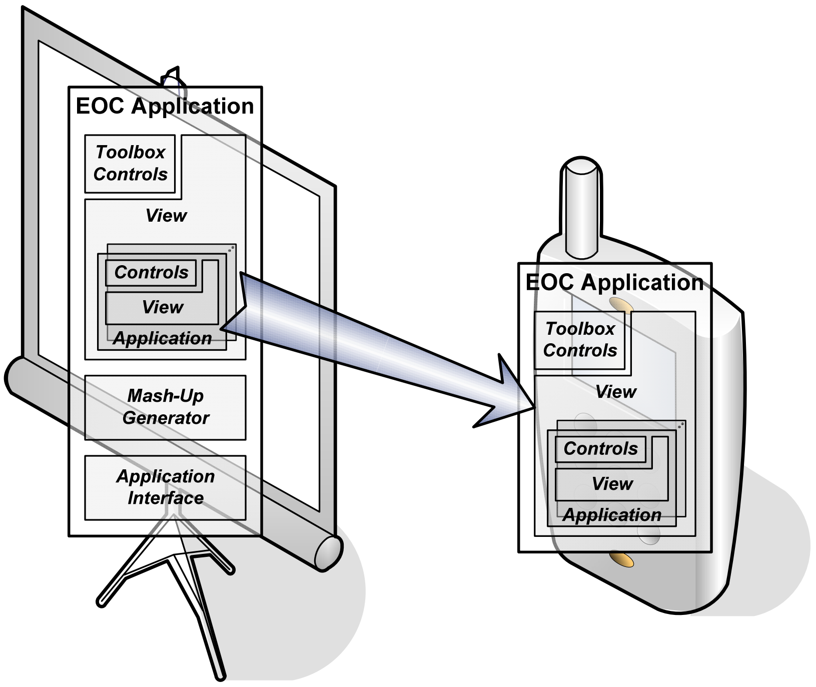


Figure 7: Migration of the EOC Application from a smart wall to a cell phone.

The business scenario from D5.1 can be generalized in yet another way, which is the use of animation. For example, rather than showing the final state of a simulation of flooding, the progress of the flood can be shown as an animation representing water rising on a map. Similarly, evacuation of people can be shown as shrinking geometric figures representing the decrease in numbers as people are evacuated. Generally speaking, the simulation of any change of parameters relevant to the business can be represented dynamically as an animation.

7.2 Challenges derived from the Generalized Scenario

The generalized business scenarios will now be used to examine the four dimensions of complexity previously identified: number of users of one device, whether a change of modality is required, number of applications involved, and how presentation and control elements of multiple applications are combined.

Taking account of these dimensions actually extends the original goal of the project, which is migration of one single-user application from one device, the source, to another device, the target. Thus, rather than taking support for these dimensions as requirements, they are better understood as challenges for on-going research.

As explained in Section 4, and shown in Figure 1 the key element in this migration is the semantic redesign process, which is a transformation process that translates a Concrete User Interface (CUI) for a specific source platform into a correspondent Concrete User Interface for a specific target platform, while preserving the semantics of the interaction as well as the current state of the interaction. This is the baseline use case and is characterized in **Table 1** according to the established dimensions of complexity.

Each of the generalized scenarios in the table is described in more detail in the following sections. Note that so far the analysis of business use cases has only discovered requirements for UI migration. Thus, the analysis concerns only UI migration as opposed to application logic migration.

Scenario	Number of users	Change of modality	Number of applications	Combination of elements
Baseline	1	possibly	1	not applicable
Mobile Audio	1 or more	yes	1 or more	possibly
Particular Mobile Audio	multiple	yes	multiple	yes
Screen Partitioning	1 or more	no	2 or more	no
Element Combination	1 or more	no	2 or more	yes

Table 1: Scenarios characterized by dimensions of complexity

7.2.1 Baseline

The standard baseline scenario is the migration of the user interface of one single-user application from the source device to the target device. As far as permitted by the device characteristics this is a complete migration of the entire UI. A variant of this scenario would be to give the user control over which elements of the UI to migrate.

In the usual case the modality of the UI would not be changed, although, again, it would be possible to permit the user to request a change of modality, e.g., from standard WIMP (window, icon, menu, pointing device) to speech and audio. Combination of elements from the UIs of multiple applications is not applicable, since only one application is involved in this scenario.

An example of this type of migration is described in D5.1 Section 3.1.1, where the UI for the flood simulation program is migrated from a desktop PC to a PDA.

7.2.2 Mobile Audio and Particular Mobile Audio Scenarios

This scenario, which corresponds to the third phase of the generalized business scenario, requires a change in modality from WIMP to audio. Adding to the complexity of this scenario, there are actually multiple users of the applications. There are the people at the planning center, manipulating the UI elements on the smart wall, and there is the manager underway, who listens in to what is being done. In other words, selected elements of the UI are duplicated in a different modality, rather than being migrated from one device to another.

A further complexity in this scenario is that the envisaged application for emergency response planning brings together multiple applications on the smart wall. Some of these applications are: various simulations and/or animations of the extent of flooding over time, various simulations of response plans, e.g., evacuation plans or sand-bagging actions, and the assignment of response forces to tasks and locations. As a result, what the manger hears may be the combination of information from multiple applications.

An example of what the manager hears compared to what UI manipulations happen should make this clearer. In the emergency scenario, available emergency crews can be assigned to evacuation points by putting a finger onto the crew on the map and then dragging towards the evacuation point. After the crew is dropped onto a point the system would request the user to enter the time interval of the planned deployment. What the manager would hear after the completion of this task would be something like:

> "John assigned Crew 10 of the Cologne Fire Brigade to evacuate Zone 10 between 10 and 11 on Monday, the 20th of January."

In fact, a task-level description of the application could aid generation of this audio message by representing this task in an abstract way, that is, the task could be modeled as:

> "<u>person</u> assigned Unit <u>x</u> of Organization <u>y</u> to evacuate Zone <u>z</u> between <u>t1</u> and <u>t2</u> on <u>weekday</u>, the <u>date</u>".

Depending on the level of detail the manager wants to hear, more information might be added to the audio message, e.g.,

> "Crew 10 has 9 members. Zone 10 is approximately 4 square kilometers in size and centers on Cologne West."

In this example, the information about the crew may come from one application and the information about the location and size of the evacuation area from others. A technique like mashups can be used to glue these applications together. See Section 7.5.4 for more information about mashups.

As summarized in **Table 1**, this particular mobile audio scenario does involve multiple users, multiple applications and the combination of UI elements along with change of modality.

There is one more important point which should not be overlooked, which is that there may be a need to identify which user performed which action at the smart wall. In fact, this was taken for granted in discussing what the manager overhears.

7.2.3 Screen Partitioning

Rather than combining UIs of multiple applications the smart wall may just be partitioned among several applications. For example, two groups of people might independently work on two different applications that have been migrated to two different halves of the smart wall. This scenario is, in essence, the same as the baseline case, except that the each application has to be assigned a separate part of the smart wall.

An example of this scenario would be that the two experts on flooding migrate (or duplicate) their respective map-based UIs to different halves of the smart wall, so that others can better see and interact with the simulations.

Note that when a smart wall is used, it is assumed that multiple people may use it simultaneously. This is enabled by the multi-touch capability of the smart wall. It should be noted that this may potentially raise synchronization issues.

7.2.4 Element Combination (Multiple Variants)

A natural extension to the previous simulation example is to combine the different flooding simulations on the same map. This might be done as follows. First, one simulation is migrated to a free smart wall. Next, the second simulation is migrated, but only the presentation elements. The control elements are merged with those of the first simulation. In this way, the simulations are synchronized, e.g. same starting and end times, same rate of running animations, etc. Note that the parameter specifying the initial data sets used for the simulations are not merged, since the whole point is to compare the results of different input datasets.

This type of sharing of results on the smart wall has two variants: what-if comparisons, and the combination of results from multiple types of experts, which is the more general case. The first involves comparing two (or possibly more) sets of comparable results from the same type of expert/program. In this case, each expert uses different input parameters or assumptions. This is actually the case described in D5.1.

The first variant requires being able to specify how to present the results so that they can be compared. The second variant requires being able to specify how to combine results that represent different aspects of a situation.

Comparison or combination on the same map is a special case, and is actually the case which is important in the specific business scenario from D5.1. This then constitutes a major requirement raised by the business scenario.

A related essential requirement is support for time-based comparison on a map (animation). This implies the need for synchronization of multiple animations as described above.

There are still other variants of combing elements. An important one for the business scenario is to migrate a simulation program to its proper place in a UI which is already being presented on the smart wall. This might be done by indicating which element, e.g., a tab area, to use for the presentation of the program to be newly migrated to the smart wall.

Another variant of element combination is the explicit specification of how to combine elements. An example is represented by specification of aggregation of information, e.g., addition of some values and display of the mean only.

This is related to the privacy issue discussed in Section 7.4.2, in that it may be permissible to make aggregated results public under some conditions. For example, statistical results of an employee survey are made public only to the extent that they do not reveal any information about individual employees.

7.3 KML

In the business application considered it seems important to exploit geographical data. For this purpose we are considering to use KML (http://www.opengeospatial.org/standards/kml). Google submitted KML (formerly Keyhole Markup Language) to the Open Geospatial Consortium (OGC) to be evolved within the OGC consensus process with the following goal: KML Version 2.2 will be an adopted OGC implementation standard. Future versions may be harmonized with relevant OGC standards that comprise the OGC standards baseline. There are four objectives for this standards work:

- That there be one international standard language for expressing geographic annotation and visualization on existing or future web-based online and mobile maps (2d) and earth browsers (3d).
- That KML be aligned with international best practices and standards, thereby enabling greater uptake and interoperability of earth browser implementations.
- That the OGC and Google will work collaboratively to ensure that the KML implementer community is properly engaged in the process and that the KML community is kept informed of progress and issues.
- That the OGC process will be used to ensure proper life-cycle management of the KML Standard, including such issues as backwards compatibility.

The OGC has developed a broad Standards Baseline. Google and the OGC believe that having KML fit within that family will encourage broader implementation and greater interoperability and sharing of earth browser content and context.

KML is an XML language focused on geographic visualization, including annotation of maps and images. Geographic visualization includes not only the presentation of graphical data on the globe, but also the control of the user's navigation in the sense of where to go and where to look.

From this perspective, KML is complementary to most of the key existing OGC standards including GML (Geography Markup Language), WFS (Web Feature Service) and WMS (Web Map Service). Currently, KML 2.2 utilizes certain geometry elements derived from GML 2.1.2. These elements include point, line string, linear ring, and polygon.

The OGC and Google have agreed that there can be additional harmonization of KML with GML (e.g. to use the same geometry representation) in the future. The Mass Market Geo Working Group (MMWG) in the OGC will establish such additional harmonization activities. OGC specifications such as Context and Styled Layer Descriptor (SLD) may be considered.

7.4 Other Considerations

In addition to the dimensions of complexity already discussed, two other things have to be taken into account in regard to the business scenario: conditions for beginning a migration and privacy issues. What they have in common is the need for a way to specify rules. One option being considered for specifying such rules is Structured English as described in the OMG Semantic Business Vocabulary and Business Rules (SBVR) specification.

7.4.1 Conditions for Beginning Migration

In phase two of the generalized scenario, in which the manager is alerted, a migration request is triggered when the stream discharge rate, one of the parameters monitored on the smart wall, exceeds a given value. Thus, there is a need to be able to specify such a condition.

7.4.2 Privacy Issues

Some data is confidential and should not be displayed on a public screen like a smart wall, unless explicitly permitted. This leads to the requirement that it must be possible to specify levels of confidentiality of data to be displayed, determine if a display is public or private and ask the user for permission to display data, if there is any doubt about violating privacy.

7.5 RIA: Implementation Choices and Consequences

In the previous section challenges related to different types of complexity were the topic. In this section, the focus is on the implementation choices made for the business scenario. The reasons for the implementation choices are explained and some of the consequences for migration are touched upon.

The major implementation decisions are to implement the business scenarios as Rich Internet Applications (RIAs) using Microsoft Silverlight and Microsoft Virtual Earth.

7.5.1 Why RIA?

Today's business users expect web applications to behave in much the same way as their desktop applications. Only RIAs effectively meet this expectation. Moreover, thanks to the RIA capability of asynchronous communication with the server, individual fields can be updated without a complete screen refresh, and the application feels more responsive.

7.5.2 Why Silverlight?

When looking at current technologies to realize Rich Internet Applications (RIAs) there are mainly two major competitors – Adobe Flex and Microsoft Silverlight.

Adobe Flex (http://www.adobe.com/de/products/flex/) is a well established framework to develop RIAs. The latest version, Adobe Flex 3.0, was released in February 2008. With the addition of LiveCycle Data Services (licence necessary) the Adobe Flex applications now are able to perform messaging, data management and PDF generation. Flex RIAs are developed using MXML, an XML based user interface markup language. In combination with ActionScript, MXML is used to declaratively describe the user interface as well as non-visible components like server side calls for the connection between user interface and backend data sources. To run Flex applications on the client the Flash plug-in needs to be installed. Adobe Flex Applications are cross-platform compatible (Windows, MAC and Linux).

Microsoft Silverlight (http://silverlight.net/) is also a cross-platform web application framework for RIAs. Like Flex it also requires installation of software in the browser. Silverlight is based on the .NET technology and the Windows Presentation Foundation (WPF). Microsoft Silverlight is, in fact, a web-based subset of WPF.

Silverlight applications are also developed using a declarative user interface description - XAML (eXtensible Application Markup Language). The acronym originally stood for Extensible Avalon Markup Language - Avalon being the code-name for Windows Presentation Foundation (the graphical subsystem feature of the .NET Framework 3.0). XAML is used as a user interface markup language to define UI elements, data binding, eventing, and other features. Silverlight applications are developed using XAML, and either JavaScript, Visual Basic, C# or Ruby for coding.

Both frameworks, Adobe Flex and Microsoft Silverlight, offer similar features like cross-platform capability, XML based user interface development and developer and design tools (Visual Studio and Expression Blend for Silverlight, FlexBuilder for Flex). Unique to Microsoft Silverlight is the support of different languages like Visual Basic, C# and Ruby. Also the distinction between user interface description and backend logic seems to be better realized with XAML than MXML.

Microsoft Silverlight, furthermore, offers a better architecture and tools for developing applications that can adapt themselves at runtime. XAML doesn't define a static user interface. Using JavaScript it is possible to adapt the XAML user interface description based on events. Thus it is possible to dynamically change the UI at runtime. These adaptations don't need a complete reload of the XAML file; only the code that changes needs to be transferred to the client.

Another important factor is the very fast growing community around Microsoft Silverlight which provides a lot of support and development examples. Using Expression Blend makes it very easy to create a cross-platform user interface with advanced adaptation capabilities.

To support mobile devices Microsoft Silverlight will provide a mobile version soon. An interesting question regarding this is whether it will provide support for the transformation of desktop interfaces to mobile interfaces.

Performance is another advantage of Microsoft Silverlight. Compared with Adobe Flex applications Microsoft Silverlight seems to be much faster in execution (see http://bubblemark.com/). Since performance and response time is a key element to realize user friendly application Microsoft Silverlight seems to be the better choice instead of Adobe Flex.

7.5.3 Why Virtual Earth?

For building Microsoft Silverlight applications that contain maps, Microsoft Virtual Earth is the best choice. There are already many examples available of how to integrate Microsoft Virtual Earth into a Silverlight application. One promising example is VIEWS (Virtual Earth Wrapper for Silverlight) (http://www.codeplex.com/views). VIEWS is a wrapper around the JavaScript Virtual Earth control which supports the development of complex Virtual Earth mashups. When developing map applications with Microsoft Silverlight the DeepZoom feature is very impressive. It allows scaling and zooming of high resolution imagery using a MultiScaleImage object.

Using Google Maps inside a Silverlight application is an alternative to Virtual Earth. But there are almost no examples or support on how to do this. Using a plain Google Maps integration is most probably not enough, since there is the need for additional controls to manipulate the map. These controls don't exist for Google Maps whereas a lot of examples are available when using Microsoft Virtual Earth.

Another unique feature of Virtual Earth (or LiveSearch) is the “birds view”. In this view the perspective is changed from a vertical to a 45 degree angle. It is possible to change the view in all four directions. Using this view enables the presentation of 3D information on the map. It will be possible to not only see the length and width of an object but also the height. This is very useful for emergency situations like the flooding scenario. Relative to the vertical view it has the advantage of being easier to interpret, since one could see the height of the water in 3D on the map.

7.5.4 What is a mashup?

By definition a mashup is the combination of existing information to create new content. To do so mashups call external sources, integrate the data from these external sources, and wire inputs from one source into outputs of another. The difficulty is to display the information from the external sources in the right way. The important component when building mashups is the APIs provided by the external sources.

If for example the task is to create a Silverlight application that puts together a Virtual Earth map and weather information this would be a mashup. The Virtual Earth map component already exists and can be integrated into the Silverlight application using the Virtual Earth API. The same applies for the weather information. There are many Web Services providing information about the weather. To integrate this information one only needs to use the interface of the Web Service. The combination of these two external information sources into one application (e.g. a weather map) is called a mashup.

It is anticipated that the business applications built in the OPEN project will use mashups as well. If for example the task is to combine the views of an application on a PDA and another application that runs on a laptop into one application that should run on a smart wall, this follows the mashup definition. The smart wall will combine information from different external data sources (the programs used by the PDA and Laptop) to build new content.

In general, the smart wall will have to provide the ability to combine several external sources in a meaningful way. To do so XAML could be used to implement the user interface but keep it dynamical enough to integrate any external sources. In other words, XAML could be used to implement a generic mashup with dynamic data sources.

8 Conclusions

In this deliverable we have briefly discussed the motivations and the state of art in the area of user interface description languages. We have shown that they can play an important role to support dynamic adaptation across various type of interactive devices because they can provide device-independent languages that can be used to create references across various possible implementations.

We have discussed how currently the user interface migration software can benefit from the use of such logical interface descriptions in XML format. Currently the user interface migration software uses TERESA XML, but we plan to move to MARIA XML language. The reason of this change is that the various case studies considered raise a number of complex issues, which showed the need for a more powerful description language. We have also started to discuss how to address issues raised by the business case study. In this case the plan is to use XAML as implementation language. Thus, if this choice will be confirmed, we can consider the possibility to design develop a transformation from MARIA XML to XAML in order to dynamically generate implementations in this language.

9 References

Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J. UIML: An Appliance-Independent XML User Interface Language, Proceedings of the 8th WWW conference, 1999.

Berti, S., Correani, F., Paternò, F., and Santoro, C. (2004). The TERESA XML language for the description of interactive systems at multiple abstraction levels, in the Proceedings of the Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages 2004, May 2004, Gallipoli, Italy, pp. 103–110. http://giove.isti.cnr. it/cameleon/cp25.html.

G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, J. Vanderdonckt. *A unifying reference framework for multi-target user interfaces*, Journal of Interacting With Computer, Elsevier Science B.V, June, 2003, Vol 15/3, pp 289-308.

Chesta, C., Paternò, F. and Santoro, C. (2004): Methods and Tools for Designing and Developing Usable Multi-Platform Interactive Applications. In Psychnology, 2 (1) pp. 123-139

Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., Lopez-Jaquero V. USIXML: A Language Supporting Multi-path Development of User Interfaces. EHCI/DS-VIS 2004: 200-220

Mori, G., Paternò, F., Santoro, C., Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. IEEE Transactions on Software Engineering (August 2004, 30,8, pp.507-520)

Nichols, J., Myers, B. A., Higgins, M., et al. (2002). Generating remote control interfaces for complex appliances, in the Proceedings of the ACM Symposium on User Interface Soft ware and Technology (UIST 2002), 27–30 October 2002, Paris, France, pp. 161–170. New York: ACM Press.

OPEN D5.1, Initial Application Requirements and Design, S. Bigi, G. Cherchi, D. Deledda, F. Mureddu, K.-U. Schmidt, October 2008

Paternò F., Model-Based Design and Evaluation of Interactive Applications, Springer Verlag, 1999.

Paternò F., Santoro C., Scorcia A.: Automatically adapting web sites for mobile access through logical descriptions and dynamic analysis of interaction resources. AVI 2008: 260-267

Szekely, P. (1996). Retrospective and challenges for modelbased interface development, in the Proceedings of the 3rd International Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'96), pp. 1–27. Vienna: Springer-Verlag.

Vanderdonckt, J. (2005). A MDA-compliant environment for developing user interfaces of information systems, in the Proceedings of the 16th

Conference on Advanced Information Systems Engineering (CAiSE 2005), pp. 16–31. Berlin/Heidelberg: Springer-Verlag.