# Reverse Engineering Cross-Modal User Interfaces for Ubiquitous Environments

Renata Bandelloni, Fabio Paternò, and Carmen Santoro

ISTI-CNR, Via G.Moruzzi, 1
56124, Pisa, Italy
{Renata.Bandelloni,Fabio.Paterno,Carmen.Santoro}@isti.cnr.it

**Abstract.** Ubiquitous environments make various types of interaction platforms available to users. There is an increasing need for automatic tools able to transform user interfaces for one platform into versions suitable for a different one. To this end, it is important to have solutions able to take user interfaces for a given platform and build the corresponding logical descriptions, which can then be manipulated to obtain versions adapted to different platforms. In this paper we present a solution to this issue that is able to reverse engineer even interfaces supporting different modalities (graphical and voice).

**Keywords:** Reverse Engineering, Cross-Modal User Interfaces, Model-based Approaches.

## 1 Introduction

In recent years, one of the main characteristics of Information and Communication Technology is the continuous proliferation of new interactive platforms available for the mass market. They vary not only in terms of screen size, but also in terms of the interaction modalities supported. Indeed, if we consider the Web, which is the most common interaction environment, we can notice that recently a number of W3C standards have been under development in order to also consider interaction modalities other than the simple graphical one.

One important consequent problem is how to obtain applications that can be accessed through such a variety of devices. It can be difficult and time-consuming to develop user interfaces for each potential platform from scratch. In order to address such issues in recent years there has been an increasing interest in model-based approaches able to allow designers to focus on the main logical aspects without having to deal with a plethora of low-level details. To this end, a number of device-independent markup languages have been proposed to represent the relevant models in device-independent languages (see for example XIML, UIML, UsiXML, TERESA XML). However, developing such model-based specifications still takes considerable effort. In order to reduce such effort there are two possible general approaches: informal-to-formal transformations or reverse engineering. In informal-to-formal approaches the basic idea is to take informal descriptions, such as graphical sketches or natural language descriptions of scenarios, and try to infer, at least partly, the corresponding logical abstractions. Reverse engineering techniques aim to obtain transformations able to

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-540-92698-6_37

J. Gulliksen et al. (Eds.): EIS 2007, LNCS 4940, pp. 285–302, 2008.

analyse implementations and derive the corresponding logical descriptions. Thus, they can be a useful step towards obtaining new versions of an implementation more suitable for different platforms.

Solutions based on syntactical transcoders (for example from HTML to WML) usually provide results with poor usability because they tend to fit the same design to platforms with substantial differences in terms of interaction resources. One possible solution to this problem is to develop reverse engineering techniques able to take the user interface of existing applications for any platform and then build the corresponding logical descriptions that can be manipulated in order to obtain user interfaces for different platforms that share the original communication goal, but are implemented taking into account the interaction resources available in the target platforms. This requires novel solutions for reverse engineering of user interfaces, given that previous work has focused only on reverse engineering of graphical desktop user interfaces.

In this paper we present ReverseAllUIs, a new method and the associated tool able to address such issues. We first provide some background information regarding the logical framework underlying this work and the various logical descriptions that are considered. We introduce the architecture of our tool, indicating its main components, their relations and describing its user interface. Then, we discuss how in our environment both vocal and graphical interfaces can be reverse engineered through a number of transformations by describing each transformation involved when considering cross-modal interfaces (interfaces of applications that can be accessed through either one modality or another one). Lastly, some conclusions along with indications for future work are provided.

## 2 Related Work

Early work in reverse engineering for user interfaces was motivated by the need to support maintenance activities aiming to re-engineer legacy systems for new versions using different user interface toolkits [9, 13], in some cases even supporting migration from character-oriented user interfaces to graphical user interfaces.

More recently, interest in user interface reverse engineering has received strong impetus from the advent of mobile technologies and the need to support multi-device applications. To this end, a good deal of work has been dedicated to user interfaces reverse engineering in order to identify corresponding meaningful abstractions [see for example 2, 3, 6, 7, 11]. Other studies have investigated how to derive the task model of an interactive application starting with the logs generated during user sessions [8]. However, this approach is limited to building descriptions of the actual past use of the interface, which is described by the logs, but it is not able to provide a general description of the tasks supported, which includes even those not considered in the logs. A different approach [5] proposes re-engineering Java graphical desktop applications to mobile devices with limited resources, without considering logical descriptions of the user interface. One of the main areas of interest has been how to recover semantic relations from Web pages. An approach based on visual cues is

presented in [15], in which semantic relations usually apply to neighbouring rectangle blocks and define larger logical rectangle blocks.

The next section discusses the various possible logical levels that can be considered for user interfaces in ubiquitous environments. Previous work in reverse engineering has addressed only one level at a time. For example, Vaquita and its successors [2, 3] have focused on creating a concrete user interface from Web pages for desktop systems. WebRevenge [11] has addressed the same types of applications in order to build only the corresponding task models.

In general, there is a lack of approaches able to address different platforms, especially involving different interaction modalities, and to build the corresponding logical descriptions at different abstraction levels: our work aims to overcome this limitation.

## 3 Background

In the research community in model-based design of user interfaces there is a consensus on what constitutes useful logical descriptions [4, 12, 14].

We provide a short summary for readers unfamiliar with them:

- The task and object level, which reflects the user view of the interactive system in terms of logical activities and objects that should be manipulated in order 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.

Thus, for example we can consider the task "select an artwork": this implies the need for a selection object at the abstract level, which indicates nothing regarding the modality in which the selection will be performed (it could be through a gesture or a vocal command or a graphical interaction). When we move to the concrete description then we have to assume a specific modality, for example the graphical modality, and indicate a specific modality-dependent interaction technique to support the interaction in question (for example, selection could be through a radio-button or a list or a drop-down menu), but nothing is indicated in terms of a specific implementation language. When we choose an implementation language we are ready to make the last transformation from the concrete description into the syntax of a specific user interface implementation language. The advantage of this type of approach is that it allows designers to focus on logical aspects and take into account the user view right from the earliest stages of the design process.

In the case of interfaces that can be accessed through different types of devices the approach has additional advantages. First of all, the task and the abstract level can be described through the same language for whatever platform we aim to address, which means through device-independent languages. Then, in our approach, TERESA XML

[1], we have a concrete interface language for each target platform. By platform we mean a set of interaction resources that share similar capabilities (for example the graphical desktop, the vocal one, the cellphone, the graphical and vocal desktop). Thus, a given platform identifies the type of interaction environment available for the user, and this clearly depends on the modalities supported by the platform itself. Actually, in our approach the concrete level is a refinement of the abstract interface depending on the associated platform. This means that all the concrete interface languages share the same structure and add concrete platform-dependent details on the possible attributes for implementing the logical interaction objects and the ways to compose them indicated in the abstract level. All languages in our approach, for any abstraction level, are defined in terms of XML in order to make them more easily manageable and allow their export/import in different tools.

Another advantage of this approach is that maintaining links among the elements in the various abstraction levels provides 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. A further advantage is that designers of multi-device interfaces do not have to learn the many details of the many possible implementation languages because the environment allows them to have full control over the design through the logical descriptions and leave the implementation to an automatic transformation from the concrete level to the target implementation language. In addition, if a new implementation language needs to be addressed, the entire structure of the environment does not change, but only the transformation from the associated concrete level to the new language has to be added. This is not difficult because the concrete level is already a detailed description of how the interface should be structured.

The purpose of the logical user interface XML-based languages is to represent the semantics of the user interface elements, which is the type of desired effect they should achieve: they should be able to allow the user to accomplish a specific basic task or to communicate some information to the user. In particular, in TERESA XML there is a classification of the possible interactors (interface elements) depending on the type of basic task supported (for example single selection, navigator, activator, …) and the ways to compose them. Indeed, the composition operators in TERESA XML are associated with the typical communication goals that designers want to achieve when they structure the interface by deciding how to put together the various elements: highlighting grouping of interface elements (*grouping*), one-to-many relations among such elements (*relation*), hierarchies in terms of importance (*hierarchy*), or specific ordering (*ordering*).

## 4 Architecture

The architecture of our tool is represented in Figure 1. It can handle multiple types of input and generate multiple types of output, which are represented by the arrows on the border of the rectangle associated with the ReversAllUIs tool.

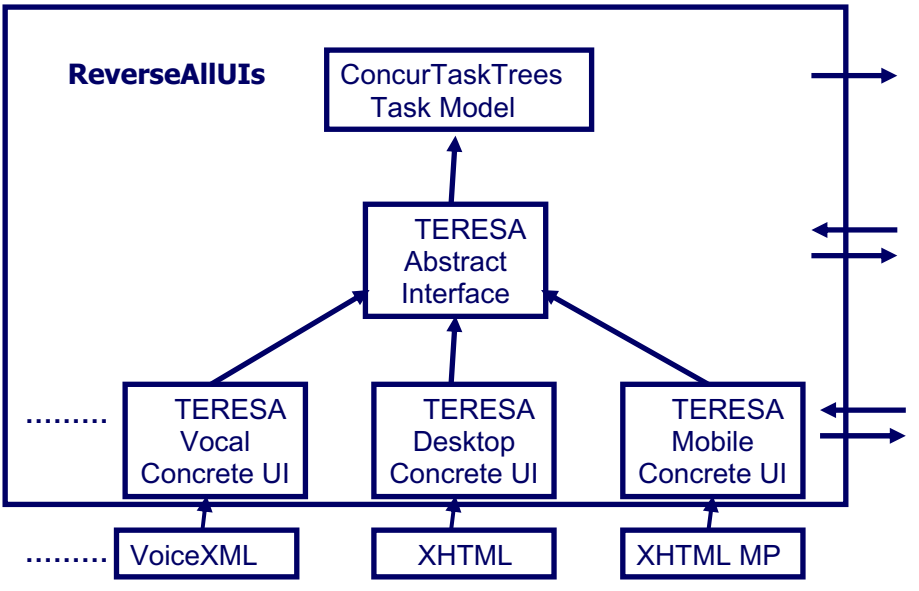


**Fig. 1.** The architecture of the tool

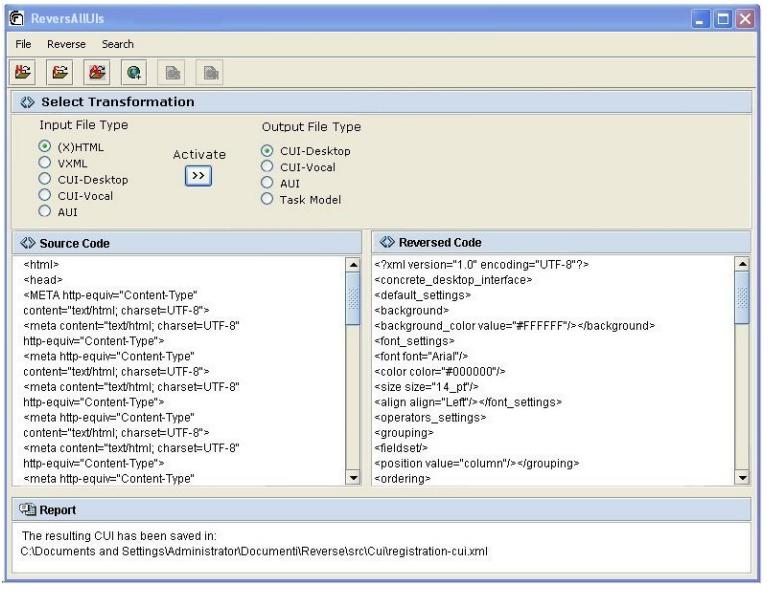


**Fig. 2.** The user interface of the tool

Our current tool addresses VoiceXML, XHTML and XHTML Mobile Profile (MP) as implementation languages, but we are planning to support additional languages (such as X+V and Java, including the version for digital TV). One main characteristic

is that the tool can receive as input not only user interface implementations, but also descriptions at intermediate abstraction levels, which can be reversed in order to obtain higher level descriptions. The highest level description is the task model, which, consequently, can only be an output for our tool.

Figure 2 shows the user interface of the tool. It allows the designer to select the type of input and output file. In the list of available input files there are implementation languages (such as XHTML and VoiceXML), concrete user interfaces (CUI) that depend on the platform (such as desktop and vocal) and the abstract specification (AUI), which is both implementation language and platform-independent.

Both the source file and the resulting reversed file are displayed. In the bottom some report messages are presented.

## 5 XHTML/CSS-to-Desktop or Mobile Concrete Descriptions Transformation

The reverse tool can reverse both single XHTML pages and whole Web sites. A Web site is reversed considering one page at a time and reversing it into a concrete presentation. Thus, the tool builds connections among the different presentations depending on the navigation structure of the Web site, and the presentations are arranged into a single concrete description representing the whole Web site.

When a single page is reversed into a presentation, its elements are reversed into different types of concrete interactors and combination of them. The reversing algorithm recursively analyses the DOM tree of the X/HTML page starting with the *body* element and going in depth. For each tag that can be directly mapped onto a concrete element, a specific function analyses the corresponding node and extracts information to generate the proper interactor or composition operator. In the event that a CSS file is associated to the analysed page, for each tag that could be affected by a style definition (such as background colour, text style, text font) the tool checks possible property definitions in the CSS file and retrieves such information to make a complete description of the corresponding concrete interactor.

Then, depending on the XHTML DOM node analysed by the recursive function, we have three basic cases:

- The XHTML element is mapped into a concrete interactor. A tag can correspond to multiple interactors (e.g. input or select tag): in this case the choice of the corresponding interactor depends on the associated type or attributes. This is a recursion endpoint. The appropriate interactor element is built and inserted into the XML-based logical description.
- The XHTML node corresponds to a composition operator, for example in the case of a div or a fieldset. The proper composition element is built and the function is called recursively on the XHTML node subtrees. The subtree analysis can return both interactor and interactor composition elements. Whichever they are, the resulting concrete nodes are appended to the composition element from which the recursive analysis started.
- The XHTML node has no direct mapping to any concrete element. If the XHTML node has no child, no action is taken and we have a recursion endpoint, otherwise recursion is applied to the element subtrees and each

child subtree is reversed and the resulting nodes are collected into a grouping composition.

Table 1 shows the main XHTML tags and the corresponding interactors/operators for the desktop concrete description.

**Table 1.** Reversing XHTML to CUI-Desktop

| X/HTML Element | CUI-Desktop Element |
|---|---|
| <ol> | OrderedList(Ordering) |
| <ul> | Bullet(Grouping) |
| <table> | Fieldset, BgColor(Grouping) DescriptionTable |
| <tr> | Fieldset, BgColor(Grouping) TableRow |
| <td> | Fieldset, BgColor(Grouping) TableData |
| <select> | List_box Drop_down_list |
| <select multiple> | ListBox |
| <textarea> | Textfield |
| <form> | Form(Relation) |
| <input type=text> | Textfield |
| <input type=checkbox> | Checkbox |
| <input type=radio> | Radiobutton |
| <input type=reset> | ResetButton |
| <input type=submit> | SubmitButton |
| <input type=button > | Button ButtonAndScript |
|  | Fieldset, Bullet, BgColor(Grouping) |
| <fieldset> | Fieldset(Grouping) |
| <a> | TextLink ImageLink mailto |
| <h1>..<h6> <b>   <i> <tt> <code> <cite> <def> <kbd> <big> <small> <sub> <sup> <var> | Textual |
| <img> | Image |

As we can see from Table 1, some of the XHTML tags can be reversed into more than one concrete element. The choice of the proper elements depends on the attributes of the XHTML tags.

In the case of the <table> tag, we considered that often it is used in order to define the layout of the page, even if it is generally considered not a good design choice. When reversing a XHTML table it is necessary to recognise the purpose for which it has been used. When it is a proper table showing data, it is reversed into the corresponding *table* concrete element, otherwise it is considered as a technique for grouping the contained elements. Some rules used to distinguish layout tables from data tables are:

- tables with attribute "border = 0" are probably layout tables,
- tables with attribute border set to a value greater than 0 are probably data tables,
- tables having tag <body> as a parent and no other sibling tags are layout tables,
- tables having the summary attribute are data tables,
- tables that define a caption element are data tables.

After the first generation step, the logical description is optimised by eliminating some unnecessary grouping operators (mainly groupings composed of one single element) that may result from the first phase. This can happen for example with tags such as  and <fieldset> that are automatically reversed into groupings but whose content includes only a single interactor, such as piece of text and images that can be joined into a single description interactor.

XHTML MP is a subset of XHTML more suitable for mobile devices. The concrete description for the mobile device platform is also a subset of that for the desktop system, it provides a smaller set of elements for implementing the higher level interactors and composition operators. Thus, when a XHTML MP implementation is found, then it is required to apply a transformation that works on a subset of input and output of the transformation previously described.

## 6 VoiceXML to Vocal Concrete Description Transformation

The basic elements of a voice application written in VoiceXML are *form(s)* and *menu(s)*. The *form* element has the same purpose as the XHTML form, that is, to collect information and pass them to a server for further processing. Thus, the VoiceXML form is reversed into a *Relation* operator like the XHTML form. Inside the form, we can find *Grouping* of interactors obtained from reversing the VoiceXML form elements for entering input. Mainly they are specified through *subdialog*, *record* and *field*.

*Subdialog* is a kind of smaller voice dialog contained in the main voice dialog, thus it is reversed into a grouping of the contained elements.

*Record* performs the registration of a vocal input from the user in an audio file format, which is reversed into a concrete *vocal_input_file* element.

*Field* is used to recognize user vocal input, not in an audio file format, but as text that can be eventually matched against a grammar specified in the VoiceXML file. The field can be reversed into a *vocal_input_text* element, in case it allows free or grammar-driven vocal input, or into a *vocal_selection* in case it contains <*option*> children nodes specifying the only possible answers among which the user can choose. The field tag can specify a grammar to restrict the range of possible vocal input from the user. Such a grammar is also retrieved and specified in the corresponding concrete element.

In addition, the Relation composition obtained as output of reversing a form can also contain a Grouping of control elements derived from reversing the VoiceXML <clear> and <submit> tags.

The second basic element of VoiceXML presentations is the *menu*. The menu is used to allow the user to navigate through the same dialogue or into a new one. Thus the menu is reverse engineered into a concrete *Ordering* of navigator elements. More specifically, this navigator can be: *enumerate_menu*, *dtmf_menu* or *message_menu*, depending on the type of VoiceXML menu.

**Table 2.** Mappings from VXML to the Vocal Concrete User Interfaces

| **VXML Tag** | **CUI-Vocal Element** |
|---|---|
| <form> | ChangeContext(Relation) |
| <block> | Insert_sound, Insert_pause, Change_volume, Keywords(Grouping) |
| <subdialog> | Insert_sound , Insert_pause, Change_volume, Keywords(Grouping) |
| <record> | VocalInputFile |
| <field> | VocalInputText<br>VocalSelection |
| <clear> | ResetCmd |
| <reset> | SubmitCmd |
| <menu> | EnumerateMenu<br>DtmfMenu<br>MessageMenu |
| <prompt> | FeedbackMessage<br>SimpleText |
| <paragraph> | SimpleText |
| #text | SimpleText |
| <link> | VocalCommand |
| <prosody volume = “+X”> | IncreaseVolume(Hierarchy) |
| <prosody volume = “-X”> | DecreaseVolume(Hierarchy) |
| <audio> | Sound |

A properly designed voice user interface includes feedback messages summarising the user activity. Each concrete interactor can define a feedback message associated to the interaction object. In order to identify the feedback messages and associate them to the proper interactor, we analyse all the messages contained in the vocal presentation. Thus, all those messages that contain a field value reading as vocal output are considered to be feedback messages of the field.

The elements described can be further composed by the Hierarchy operator in the event that an increase or decrease of the vocal volume is detected. Another composition of elements is identified when different VoiceXML interface elements are enclosed between a starting and ending sound, in this case a Grouping structure is associated with the interactors corresponding to the enclosed VoiceXML elements.

**Table 3.** Mappings of CUI-Desktop and CUI-Vocal elements to Abstract elements

| CUI-Desktop | CUI-Vocal | Abstract Interface |
|---|---|---|
| OrderedList | alphabeticalOrder<br>Keywords | Ordering |
| BiggerFont | IncreaseVolume<br>DecreaseVolume | Hierarchy |
| Form | ChangeContext | Relation |
| Fieldset<br>Bullet<br>BgColor<br>Bullet | InsertSound InsertPause<br>ChangeVolume<br>Keywords | Grouping |
| RadioButton<br>ListBox<br>DropDownList | VocalSelection | SelectionSingle |
| CheckBox<br>ListBox | VocalSelection | SelectionMultiple |
| Textfield | VocalInputText | TextEdit |
| Textfield | VocalInputText | NumericalEdit |
| NOT SUPPORTED | VocalInputFile | ObjectEdit |
| ImageMap | NOT SUPPORTED | PositionEdit |
| TextLink<br>ImageLink<br>Button | VocalCommand<br>EnumerateMenu<br>DtmfMenu MessageMenu | Navigator |
| ResetButton<br>ButtonScript<br>MailTo | ResetCmd<br>SubmitCmd<br>CmdAndScript | Activator |
| SimpleText<br>TextFile | SimpleText TextFile<br>AudioFile | Text |
| Image | Sound | Object |
| TextImage<br>Table | VocalDescription | Description |
| NOT SUPPORTED | FeedbackMessage | Feedback |

## 7 Concrete Descriptions to Abstract Description Transformation

The Abstract User Interface is a platform- and implementation language-independent description of the user interface, conversely to the Concrete User Interface, which is a language specific for each platform for which the user interface is designed. This means that reversing any platform-specific concrete description yields an abstract description always in the same language. Since in TERESA XML the concrete descriptions are a refinement of the abstract one, they add implementation details to the higher level interactors defined in the abstract descriptions. The process for reversing a concrete description into the corresponding abstract one is quite simple, since it simply consists in removing the lower level details from the interactor and composition operators specification, while the structure of the presentations and the connections among presentations remain unchanged.

## 8 Example Applications

Figure 3 shows an example of a XHTML page for the desktop platform. It allows the user to navigate among different pages through a navigation menu on the left and shows a form that can be filled in and submitted for registering to a User Interface Workshop event. As you can note, when the registration page is visualised, the related link in the navigation menu on the left does not visualise “Registration” as a link.

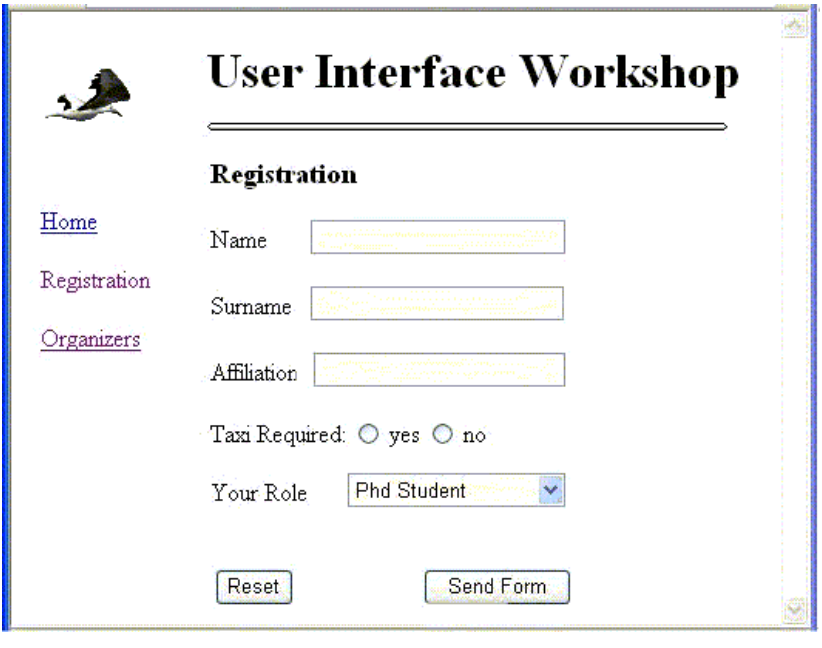


**Fig. 3.** Desktop XHTML example page

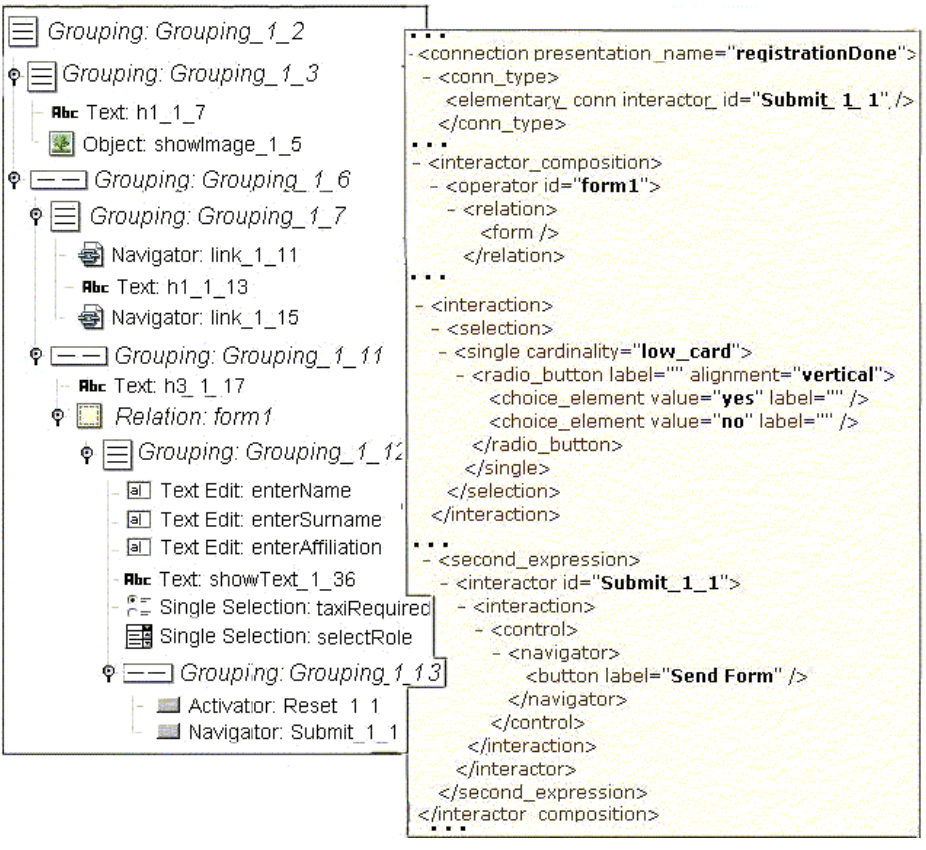


**Fig. 4.** An abstract description of the graphical example (left) and an excerpt of the corresponding XML concrete description (right)

Figure 4 shows the result of reversing the XHTML page into a Concrete User Interface. The representation of the Concrete User Interface has been obtained by loading the resulting CUI-Desktop specification in the TERESA tool, which shows in the tree-like format the higher level information (AUI level). The same figure also shows an excerpt of the XML specification of the concrete user interface obtained. Comparing the XHTML page shown in Figure 3 and the corresponding logical description shown in Figure 4 we can see that the reverse engineering of the page generates a main column grouping (Grouping_1_2) of two main groupings: Grouping_1_3 composing the interactors corresponding to the image and text at the top of the page and Grouping_1_6 containing two further compositions: Grouping_1_7 collects the interactors obtained from reversing the links of the navigation menu on the left of the page, while Grouping_1_11 composes the text introducing the form and the *Relation* that contains all the interactors corresponding to the form elements collected in Grouping_1_12. The form commands submit and reset have been reversed into the corresponding activators and collected into Grouping_1_13. In the XML specification shown on the right side of Figure 4 we can

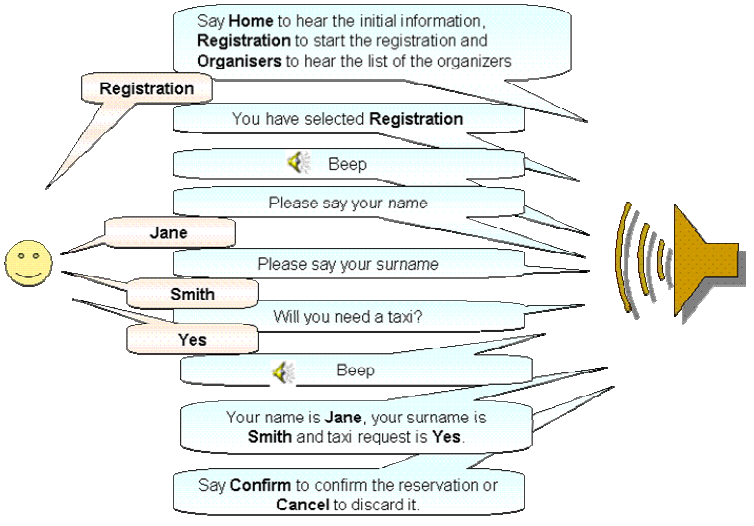


**Fig. 5.** Vocal VXML interface example

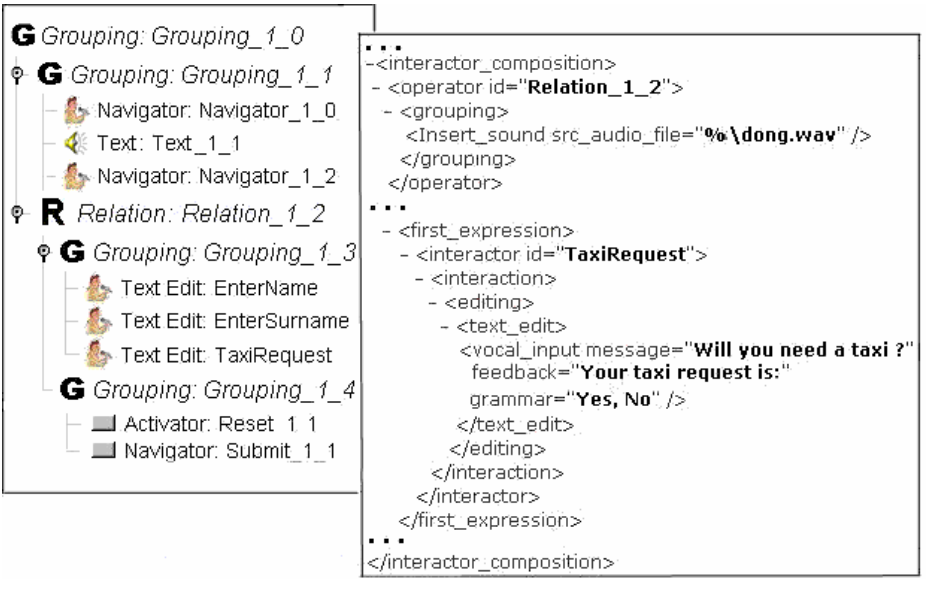


**Fig. 6.** An abstract description of the vocal example (left) and an excerpt of the corresponding XML concrete description (right)

see excerpts of the corresponding concrete description. In particular, the XML shows how the Relation composition operator is implemented by a form (*form1*) and it is connected to a *registrationDone* presentation through the *Submit1_1* button, together with the concrete interface details of the SingleSelection element named *taxiRequired*.

Figure 5 shows the vocal VoiceXML version of the simple application considered. The vocal interface starts by asking the user which dialogue to start with. Then, it accesses the registration dialogue as requested and continues to prompt for information to fill in the vocal form and then submit it. Figure 6 provides a representation of the result obtained by reversing the VoiceXML application into a Concrete User Interface. The tree view shows the abstract elements, while the XML code excerpt shows the lower level concrete details. For example, the figure shows how the part of the dialogue delimited by the two "Beep" sounds has been reversed into the grouping Grouping_1_3. The concrete interface can implement the grouping operator in different ways (see Table 2, *subdialog* element), in this case we see that the "insert sound" option has been recognized. In Figure 6 we can also see an excerpt of the concrete specification concerning the part of the dialogue that prompts for the taxi option. The XML excerpt shows the message that the vocal interface uses both for prompting and for giving feedback to the user. Moreover, it also supports the recognition of the grammar associated to this particular vocal field.

## 9 Abstract Description to Task Model Transformation

The task models that we consider are specified in the ConcurTaskTrees (CTT) notation [12], which describes them in a hierarchical format with various temporal relations that can be indicated among tasks. In addition, a number of attributes can be specified for each task. A CTT task is characterised by its "category" and "type".

The category indicates how the task performance is allocated and can take the following values:

- *Abstraction*: for higher level tasks with subtasks that do not have the same type of allocation. This category of task is associated with composition operator elements in the logical interface specification and therefore, it might be associated to the overall access to one presentation.
- *Interaction*: for tasks obtained by reversing interaction interactor elements.
- *Application*: for tasks obtained by reversing only-output interactor elements.

The root node of the task model is an abstraction task representing the whole application. As the whole application is generally composed of several presentations, the ReverseAllUIs tool starts building the task model associated to each presentation. Each presentation of the Abstract User Interface can contain both elements that are elementary interactor objects or composition operator elements.

The composition operators can contain both simple interactors and, in turn, multiple composition operators. Each composition operator in the logical user interface is reversed into an abstract task node, whose children are the tasks obtained by reversing the elements to which the abstract composition operator applies. The reversed children are connected through CTT temporal operators depending on the type of composition operator, as indicated in Table 4. For instance, if there are several objects in the same presentation and no constrain is put on the sequence about how the user is expected to interact with the different objects in the presentation, this behaviour will be translated by means of a concurrent CTT operator, which models the possibility of interacting in any order with the different objects.

**Table 4.** Reversing CUI composition operators into CTT temporal operators

| Abstract Composition Operator | CTT Temporal Operator |
|---|---|
| Grouping | Concurrency |
| Ordering | SequentialEnabling |
| Hierarchy | SequentialEnabling or Concurrency |
| Relation | Concurrency (among elements contained in the <first_expression> tag)<br><br>Disabled by (elements contained in <second_expression> tag) |

Each elementary interactor is reversed into a CTT basic task, whose category is identified through the rules explained before. Further rules exist for reversing elementary interactors. For instance, if an interactor supports an activity that might be or not carried out by the user, then such interactor will be reversed onto an optional task. Also, elementary abstract interactors can be mapped onto elementary tasks by considering the type of activity supported by the task, which can be specified with CTT notation: for instance, a text_edit AUI object will be mapped onto an interaction task having "*Edit*" as its type of activity. As a particularly interesting case of elementary interactor we consider the reverse engineering of navigators, which are objects allowing moving from one presentation to another one, and therefore, their reverse engineering involves both the presentation to which the navigator belongs and the presentations that it is possible to reach through it. The basic rule that has been identified for reverse engineering navigators is that elementary interaction tasks corresponding to navigators can disable the set of tasks associated with the current presentation and enable the next presentation. Once all single presentations have been reversed, the corresponding CTT subtrees must be composed to build up the whole application task model. The presentation subtrees are inserted, directly or grouped through a further abstraction node, as children of the root.

We describe how to reverse navigators by considering the example page considered in Figure 3. From the point of view of the abstract user interface such presentation can be seen as a presentation P1 connected to more than one presentation. Referring to Figure 7, such presentations are respectively reversed into the abstract tasks *Access Form Results*, *Access Home Page*, *Access Organisers Page*. Then, the latter presentations can be accessed by means of navigators which are reversed into corresponding interaction tasks, in our example they respectively correspond to *Select Send Form*, *Select Home*, *Select Organisers*. The fact that through navigators it is possible to reach the corresponding different presentations is modelled by connecting such tasks to the correspondingly related abstract tasks through a *SequentialEnabling* operator (represented by the >> symbol), and forming in turn three higher level abstract tasks, which in our case correspond to *Send Form*,

*Access Home* and *Access Organisers* tasks. Such tasks will be in turn connected each other through a *Choice* temporal operator (represented by the [] symbol, see Figure 7) to model the fact that the user can select only one of these paths. The abstract task obtained by such composition is in turn connected through a disabling operator with the subtree derived by reverse engineering the other elements belonging to the presentation. The disabling operator models the fact that when the user selects the navigation to a different page, it will disable the other elements in the presentation.

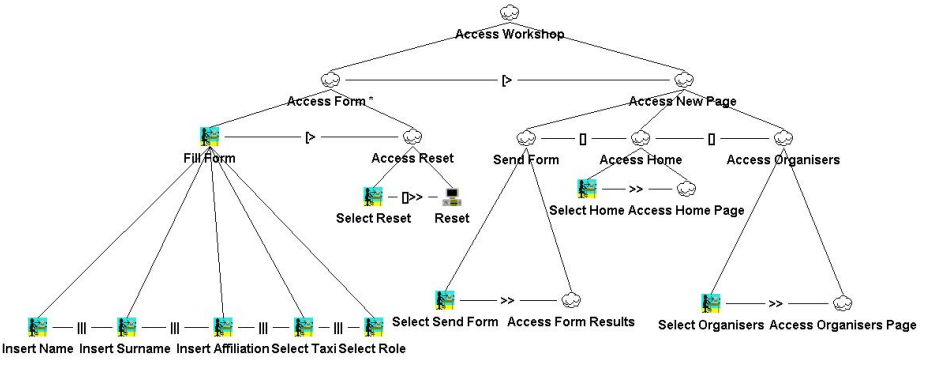


**Fig. 7.** Reverse engineering of multiple connections

As another type of navigator we consider the case when a presentation contains at least one link to a page external to the current application: in the task model an interactive task, called *Select External Link*, is added as subtask of the node grouping all the subtasks obtained by reverse engineering the whole application, which indicates that at this point the user leaves the application.

The recursive rules used in reversing the abstract logical description into the corresponding task model can generate task models with more nodes than what is strictly required. It may happen to find out *abstraction* tasks having only one child. In this case, the abstract task is removed and the child node is raised one level up. The CTT description language requires specifying the parent and sibling nodes for each task, hence, while removing a task from the tree and replacing it with its child, relationships among nodes must be updated.

## 10 Application of Reverse Engineering in Ubiquitous Environments

Reverse engineering can be used to support semantic redesign. In semantic redesign the basic idea is to transform the logical specification for a platform into a logical specification for a different one according to a number of design criteria.

Another useful application of reverse (and forward) engineering, combined with semantic redesign is the generation of migratory user interfaces. They are interfaces that can migrate among different devices while adapting to the characteristics of the

target platform and maintaining task continuity, so that the users have not to restart from scratch their activity when they change the device after a migration request.

We have developed a migration environment based on a proxy/migration server to which users have to subscribe for accessing Web applications through it. If the interaction platform used is different from the desktop, the server transforms the considered page by building the corresponding abstract description and using it as a starting point for creating the implementation adapted for the device accessing it. Also, in order to support task continuity, when a request of migration to another device is triggered, the environment detects the state of the application modified by the user input (elements selected, data entered, …) and identifies the last element accessed in the source device. Then, a version of the interface for the target device is generated, the state detected in the source device version is associated with the target device version so that the selection performed and the data entered are not lost. Lastly, the user interface version for the target device is activated at the point supporting the last basic task performed in the initial device.

## 11 Conclusions and Future Work

In the paper we have presented the ReverseAllUIs environment supporting reverse engineering of user interfaces for different platforms and modalities (graphical and voice).

These features make the tool useful in ubiquitous environments, which are characterised by the presence of various types of interaction platforms.

The logical descriptions obtained in this way can be used for many purposes. One typical use is to exploit them in order to obtain user interfaces for different platforms by exploiting the semantic information reconstructed in order to obtain more meaningful results (through semantic redesign [10]) when deriving implementations for different target platforms. The task models obtained can also be used to support usability evaluation.

Future work will be dedicated to further increasing the number of interactive platforms and modalities supported by the reverse engineering tool. We also plan to develop a Web user interface of the reverse engineering tool so that users can access it remotely, indicate the URL of a web site and receive back the specification of the corresponding logical abstractions requested.

## References

1. Berti, S., Correani, F., Paternò, F., Santoro, C.: The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Leveles. In: Proceedings Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, pp. 103–110 (May 2004)
2. Bouillon, L., Vanderdonckt, J.: Retargeting Web Pages to other Computing Platforms. In: Proceedings of IEEE 9th Working Conference on Reverse Engineering WCRE 2002, Richmond, 29 October-1 November 2002, pp. 339–348. IEEE Computer Society Press, Los Alamitos (2002)

3. Bouillon, L., Vanderdonckt, J., Chieu Chow, K.: Flexible Re-engineering of Web Sites. In: Proceedings of the International conference on Intelligent User Interfaces (IUI 2004), Madeira, pp. 132–139. ACM Press, New York (2004)
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15(3), 289–308 (2003)
5. Canfora, G., Di Santo, G., Zimeo, E.: Toward Seamless Migration of Java AWT-Based Applications to Personal Wireless Devices. In: Proceedings WCRE 2004, pp. 1–9 (2004)
6. El-Ramly, M., Ingiski, P., Stroulia, E., Sorenson, P., Matichuk, B.: Modeling the System-User Dialog using Interaction Traces. In: Proc. of the eighth Working Conference on Reverse Engineering, Stuttgart, Germany, 2-5 October 2001, pp. 208–217. IEEE Computer Soc. Press, Los Alamitos (2001)
7. Gaeremynck, Y., Bergman, L.D., Lau, T.: MORE for less: model recovery from visual interfaces for multi-device application design. In: Proceedings of the international conference on Intelligent user interfaces, Miami, Florida, USA, January 2003, pp. 69–76. ACM Press, New York (2003)
8. Hudson, S., John, B., Knudsen, K., Byrne, M.: A Tool for Creating Predictive Performance Models from User Interface Demonstrations. In: Proceedings UIST 1999, pp. 93–102. ACM Press, New York (1999)
9. Moore, M.M.: Representation Issues for Reengineering Interactive Systems. ACM Computing Surveys Special issue: position statements on strategic directions in computing research 28(4), article # 199 (December 1996)
10. Mori, G., Paternò, F.: Automatic semantic platform-dependent redesign. In: Proceedings Smart Objects and Ambient Intelligence 2005, Grenoble, pp. 177–182 (October 2005)
11. Paganelli, L., Paternò, F.: Automatic Reconstruction of the Underlying Interaction Design of Web Applications. In: Proceedings Fourteenth International Conference on Software Engineering and Knowledge Engineering, pp. 439–445. ACM Press, Ischia (2002)
12. Paternò, F.: Model-based design and evaluation of interactive applications. Springer, Heidelberg (1999)
13. Stroulia, E., Kapoor, R.V.: Reverse Engineering Interaction Plans for Legacy Interface Migration. In: Proceedings of CADUI 2002, pp. 295–310 (2002)
14. Szekely, P.: Retrospective and Challenges for Model-Based Interface Development. In: 2nd International Workshop on Computer-Aided Design of User Interfaces. Namur University Press, Namur (1996)
15. Xiang, P., Shi, Y.: Recovering semantic relations from web pages based on visual cues. In: Proceedings of the 11th international conference on Intelligent user interfaces, Sydney, Australia, January 29-February 01, 2006, pp. 342–344 (2006)