Authoring pervasive multimodal user interfaces

Fabio Paternò*, Carmen Santoro,
Jani Mäntyjärvi, Giulio Mori and
Sandro Sansone

ISTI-CNR

Via G. Moruzzi, 1

56124 Pisa, Italy

E-mail: fabio.paterno@isti.cnr.it

E-mail: carmen.santoro@isti.cnr.it

E-mail: jani.mantylarvi@isti.cnr.it

E-mail: giulio.mori@isti.cnr.it

E-mail: sandro.sansone@isti.cnr.it

*Corresponding author

Abstract: In this paper, we present an environment for authoring pervasive multimodal user interfaces. It is composed of a set of XML-based languages, transformations among such languages, and an authoring tool. It provides designers with the possibility of designing interfaces for a wide set of platforms, which support various modalities. We describe how the environment has deeply changed from the initial mono-modal, web-oriented environment and provide example applications for a number of platforms.

Keywords: model-based design of interactive applications; multimodal interfaces; pervasive environments.

Reference to this paper should be made as follows: Paternò, F., Santoro, C., Mäntyjärvi, J., Mori, G. and Sansone, S. (2008) 'Authoring pervasive multimodal user interfaces', *Int. J. Web Engineering and Technology*, Vol. 4, No. 2, pp.235–261.

Biographical notes: Fabio Paternò received his Laurea Degree in Computer Science from the University of Pisa (Italy) and his PhD from the University of York (UK). Since 1986, he has been working at C.N.R. in Pisa, where he is the Research Director and Head of the Laboratory on Human Interfaces in Information Systems at ISTI. He has been the Scientific Coordinator of several EU projects. His current research interests include migratory interfaces, methods and tools for multimodal user interface design and evaluation, user interfaces for mobile devices, model-based design of interactive systems and end-user development. He has published over 140 papers in refereed international conferences or journals.

During the last ten years, Dr. Carmen Santoro has worked on methods and tools for the analysis, design and development of interactive applications, and methods and tools for automatic support for usability evaluation. She has published papers in international conferences and journals on HCI and has been a member of several international HCI conferences. She has also been a Reviewer for international HCI journals. She was the Organisational Overviews Co-Chair of the *INTERACT 2005 Conference*, and Workshop and Tutorial

Chair of the *Mobile HCI 2002 Symposium*. She is currently Co-Chair for the *CHI 2008 Workshops*, and member of the programme committee of *INTERACT 2007*, *Tamodia 2007*, *EIS 2007* and *Mobile HCI 2007*.

Jani Mäntyjärvi received his MSc degree in Biophysics and PhD degree in Information Processing from the University of Oulu in 1999 and 2004, respectively. He is a Senior Research Scientist at the VTT Technical Research Centre of Finland in Oulu, Finland. He is a Docent in Computer Engineering (with specialty in adaptive user interaction techniques) in the Faculty of Technology at the University of Oulu. His current professional interests include technologies for adaptive interaction for mobile and pervasive computing devices.

Giulio Mori received the University Degree in Informatics Engineering from the University of Pisa and is a Research Assistant at the Laboratory on Human Interfaces in Information Systems at ISTI-C.N.R. Since end of 1999, Mori has been working on the design and development of interactive applications for different platforms.

Sandro Sansone obtained the University Degree in Computer Science on July 2006, with a thesis entitled 'Designing and automatic tv digital user interface generation on MHP platform' developed in the HIIS laboratory of ISTI-CNR in Pisa. He currently works at the R&D department of ION Trading System taking particular attention for user interface applications. He is still interested in digital television applications.

1 Introduction

In recent years an increasing number of interactive devices has been made available in the mass market (cellphones, PDAs, desktops, large screens and so on). This has raised a number of issues for designers and developers of interactive applications in order to manage such increasing complexity.

To this end, model-based approaches for interactive applications (Szekely, 1996; Paternò, 1999) have been considered as particularly useful solutions because they can identify logical descriptions that contain semantic information, which highlight the main aspects to consider and then can be transformed into a number of implementation languages. Such logical descriptions are usually described using XML-based languages (see for example, UIML (Abrams *et al.*, 1999), XIML (Puerta and Einsenstein, 2001), USIXML (Stanciulescu *et al.*, 2005), TERESAXML (Berti *et al.*, 2004)).

In this paper, we present the new multimodal TERESA environment and its possibilities. In particular, after briefly recalling the first tool and discussing its limitations, we present how such issues have been solved with this new environment, which is able to support user interface generation for a wide set of implementation languages: XHTML MP, VoiceXML, X+V, SVG, Xlet, gesture library for MS. This means that designers and developers have a consistent environment that allows them to obtain applications for a variety of platforms supporting various modalities. By *platform* we mean a set of interaction devices that share similar capabilities (examples of different platforms are: the graphical desktop, the vocal one, the cellphone, the graphical and vocal desktop, graphical and gestural, digital TV, *etc.*). 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. We also show example applications for various target platforms. Lastly, we draw some conclusions and provide indications for future work.

2 Related work

While early work in model-based user interface development (Sukaviriya and Foley, 1993) focused mainly on graphical desktop applications, in recent years such approaches have shown to be useful for designing multidevice, multimodal-user interfaces. Obrenovic *et al.* (2004) have investigated the use of conceptual models expressed in UML in order to derive graphical, form-based interfaces for desktop or mobile devices or vocal devices. UML is a software engineering standard mainly developed for modelling structure and behaviour of software applications, with limited attention to their user interface. The ICO formalism for user interfaces has shown to be suitable to model and specify multimodal interfaces mainly for analysis in safety-critical applications (Bastide *et al.*, 2004), and it has limited support for generation of multimodal interfaces from such specifications.

Several Interfaces, Single Logic (Sisl) (Ball *et al.*, 2000) is an approach for designing and implementing interactive services with multiple-user interfaces. A key idea underlying Sisl is that all user interfaces to a service share the same service logic, which provides a high-level abstraction of the service/user interaction. There are two pieces to the Sisl approach:

- 1 a standard language-independent architecture
- 2 reactive constraint graphs, a Domain-Specific Language (DSL) for designing and implementing interactive services based on an analysis of the features required of a service logic that is shared across many different user interfaces.

One of the TERESA main points, instead, is that the functionalities provided within the different user interfaces may, depending on the specific device at hand, even radically change both in terms of which services are provided and in terms of how such services are provided.

One interesting effort to ease multimodal interface development is ICARE (Bouchet *et al.*, 2004): it provides a graphical environment for a component-based user interface exploiting various modalities and modules that allow assorted compositions of such modalities. In this paper, we present a different approach: we show how we can derive multimodal interfaces starting with logical descriptions of tasks and user interfaces obtained through general descriptions of the user interface which, at the abstract level, do not bring any specific reference to a particular platform (in this sense, such descriptions, at the abstract level, can be defined as ‘platform-independent’). Such descriptions are progressively refined and made more concrete in the next steps, until the last transformation which will deliver the final user interfaces. We still provide the possibility of combining the modalities in various ways, but at different granularity levels (inside a single interaction object and among several interaction objects). While some other work has been carried out to apply transformations to logical descriptions to derive multimodal interfaces (Stanciulescu *et al.*, 2005), our work has been able to provide an

authoring environment that is able to suggest solutions for identifying how to combine various modalities and allows designers to easily modify them in order to tailor the interface generation to specific needs. This result has been obtained by extending a previously existing authoring tool (Mori *et al.*, 2004), which was limited to creating only graphical or vocal interfaces. During a first usability evaluation study conducted in an industrial setting, the tool was acknowledged to have several promising features for improving the usability both from the point of view of the designers and from the point of view of the final users, for instance in terms of the improved consistency of the design amongst different platforms (Chesta *et al.*, 2004). Moreover, in this evaluation exercise it was also found that the tool-supported methodology offers a very good support to fast prototyping, producing a first version of the interface in a significantly shorter time, while rework time results increased. The latter aspect, mainly owing to the greater familiarity of the subjects with traditional techniques than with model-based techniques and notations, is then expected to consistently be reduced by a continuous use of the tool in the software production process, thus confirming the advantages of the proposed methodology.

3 Background

In this section, we provide an introduction to the TERESA approach and its initial environment in order to provide readers who do not know it with the relevant background.

It is worth pointing out that, since TERESA is aimed at facilitating the activities of the designers, one of the key aspects of TERESA is to provide great flexibility in adequately supporting them, by providing the tool with different levels of automation. Indeed, the tool is able to provide different solutions, ranging from completely automatic solutions (suitable for novice designers) to highly interactive ones where more expert designers can tailor or even radically change the solutions proposed by the tool. Moreover, the user interfaces generated by the tool can also be specified at different abstraction levels, and it is also an additional help for the designers in that they provide different ‘views’ of the same user interface: the selection of the most appropriate view is done by the designers depending on the specific aspects they are currently interested in.

In the next section, we will provide further details about such different logical descriptions handled by the tool.

The different levels manipulated by TERESA

In the research community in model-based design of user interfaces there is a general consensus on what the useful logical descriptions are (Calvary *et al.*, 2003; Paternò, 1999; Szekely, 1996), and we followed this structure for the levels manipulated by the TERESA tool:

- 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.

Thus, for example, we can consider the task ‘switch the light on’: 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 switch or a vocal command or a graphical interaction). When we move to the concrete description, then we have to assume a specific platform, for example, the graphical PDA, 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 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 the logical aspects and take into account the user’s 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: for instance, the task and the abstract level can be described through the same language for whatever platform we aim to address, while in our approach, we have a concrete interface language for each target platform considered.

Following an ideal top-down approach, the starting point is the task level, in which the activities that should be supported by the system are specified in a hierarchical manner (in our case we use the CTT notation (Paternò, 1999)) and also expressing the temporal relationships occurring between the different tasks.

Then, at the abstract level we introduce a number of basic elements able to support different activities in a platform-independent manner: for instance we just indicate the type of activity to be performed (*e.g.*, selection, editing, *etc.*) without any reference to concrete ways to support such a performance (*e.g.*, selecting an object through a radio button or a pull-down menu, *etc.*). At this level we also describe how to compose such basic elements through some composition operators. Such operators can involve one or two expressions, each of them can be composed of one or several interactors or, in turn, compositions of interactors. In particular, the composition operators have been defined taking into account the type of communication effects that designers aim to achieve when they create a presentation (Mullet and Sano, 1995). They are:

- Grouping – indicates a set of interface elements logically connected to each other.
- Relation – highlights a one-to-many relation among some elements, one element has some effects on a set of elements.
- Ordering – some kind of ordering among a set of elements can be highlighted.
- Hierarchy – different levels of importance can be defined among a set of elements.

Therefore, an abstract user interface is composed of a number of presentations and connections among them. While each presentation defines a set of interaction techniques perceivable by the user at a given time, the connections define the dynamic behaviour of the user interface, by indicating what interactions trigger a change of presentation and what the next presentation is. Both the static arrangement of interactions in the same presentation and the dynamic behaviour of the abstract user interface are derived by

analysing the semantic of the temporal operators included in the task model specification. For instance, a sequential operator between two tasks implies that the related presentations will be sequentially triggered: this will be rendered, at the abstract level, by associating a connection between two different abstract presentations (with each presentation supporting the performance of just one task), so that the performance of the first task will trigger the activation of the second presentation and render the sequential ordering. On the contrary, a concurrency operator between two tasks implies that the associated interactors will be presented at the same time so as to support the concurrency between the connected activities, therefore the abstract objects supporting their performance will be included in the same presentation.

It is also worth noting that, at the abstract level, some work has also been done regarding the issue of how to connect the interactive part of a software application with the functional core, namely, the set of application functionalities independent of the media and the interaction techniques used to interact with the user. This aspect is very relevant in addressing the problem of generating dynamic pages with our tool. A first solution exploiting the information contained in the task model (*e.g.*, by considering the activities completely performed by the application and not handling perceivable objects in the user interface) has already been identified for this problem.

The concrete level is a refinement of the abstract 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. Figure 1 shows how an interactor of the concrete user interface can be supported on a graphical desktop platform. The elementary concrete elements are obtained as a refinement of the abstract ones and are highlighted through a different colour at the bottom level of the hierarchy shown in the figure. As you can see, a navigator 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 list. The same holds for the operators: indeed, the desktop environment allows using tables, so the grouping operator can be refined at the concrete level by a number of techniques including both unordered lists by row and unordered list by column (apart from classical grouping techniques such as fieldsets, bullets and colours). The small capability of a mobile phone does not allow implementing the grouping operator by using an unordered list of elements by column, then this technique is not available on this platform. In a vocal device, a grouping effect can be achieved through inserting specific sounds or pauses or using a specific volume or keywords (Berti and Paternò, 2003).

Figure 1 An excerpt from the concrete user interface for the graphical desktop

One advantage of this approach based on multiple levels of abstraction is that all the concrete interface languages share the same structure and add concrete platform-dependent details to the abstract language on the possible attributes for implementing the logical interaction objects and the ways to compose them. 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 allows 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 multidevice interfaces do not have to learn all the 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; 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.

4 Multimodal TERESA

The first versions of TERESA provided a convenient support for the most common form-based user interfaces. However, we soon realised that they suffered from some important limitations. For instance, no support was offered for the generation of vectorial graphics, or mixed (including both form-based and graphical elements) interfaces; neither did it enable the definition of the interactive behaviour of direct-manipulation interfaces. In addition, the need for addressing multiple modalities at a time was becoming important, since the technological evolution is making multimodal technology available to the mass market with increased reliability. Supporting various modalities is also important to obtain environments in which users can interact naturally. Moreover, the need of supporting other, less ‘traditional’ devices and modalities (such as digital TV and gesture-based modality) was seen as a meaningful test-bed for assessing the validity of the approach, and to what extent it is able to cope with the challenges related to developing, for example, gesture-based interactions on mobile devices.

In particular, we identified a set of basic requirements driving the design and implementation of the new environment:

- Not only web applications

Web applications have become very popular because of the ubiquity of the client browsers, but other technological platforms (*e.g.*, Java, .NET, *etc.*) also require appropriate attention owing to their growing diffusion. In recognition of this need, we decided to provide, in the new tool, appropriate support also for this kind of implementation environments, in order to improve the coverage offered by the tool (see Figure 2).

- **Multimodality**

Because of the increasing trend in accessing information using different types of modalities, we recognised the need for designers to exploit the use of various modalities, together with handling the specific issues connected with the use of multiple modalities. The approach followed is also suitable for paving the way for an easy inclusion of further modalities not developed yet.

- **Not forcing one specific methodology**

The tool does not force the use of one particular methodology; instead, its intended goal is to create an open environment to be opportunistically and flexibly used by designers according to the specific needs of the system considered and the ability to fit different design methodologies. Moreover, it is also worth pointing out the additional benefit that, in our tool, the selection of the most appropriate entry (and exit) points is completely left to the designers who can conveniently fit the (XML-based) solutions provided by the tool in their own methodology/framework/process.

Figure 2 The different platforms addressed by multimodal TERESA

4.1 The authoring environment

There can be different starting points with multimodal TERESA. The designer can start with a task model specified by the CTT notation, which can be transformed, through various degrees of automation, in the corresponding logical interface. Alternatively, the designer can start editing the logical interface directly. In either case, the designer can work through the environment shown in Figure 3. It is divided into four main parts: the top-left part shows the list of presentations edited so far; in the top-right part there is the abstract description of the currently selected presentation; the bottom-right part is dedicated to the concrete refinement for the current platform of the abstract element selected in the top-right part, together with the possible alternatives for the various

attributes. Lastly, in the bottom-left part there is the list of connections associated with the current presentation showing what presentations can be reached from it and through the execution of which interactor.

Figure 3 The correspondence between the content in the authoring environment (left part) and the user interface generated (right part)

The interface visualised in Figure 3 (left part) represents the multimodal TERESA authoring tool showing the user interface at the abstract/concrete levels for a movie application. We can see that the abstract part is structured through an expression of one main grouping at the highest level and three nested grouping expressions, namely, one operator groups a text with a description element; another one groups two texts and two description elements; the last grouping refers to the two navigational elements, indicating that they should be lined up horizontally. The main grouping element puts together all the groupings using (see right-bottom part of the authoring tool visualised in Figure 3) a fieldset, which is usually implemented through a rectangle, to group all the involved elements, and indicating that they should be organised vertically (as you can see from the 'column' attribute value visualised as current setting value for the element positions). Moreover, for reader's convenience, in Figure 3 we highlighted the correspondences between the abstract/concrete descriptions handled in the tool and the associated final user interface elements automatically generated from such logical descriptions. In Section 6, we will consider again this movie application example – which manages both graphical and vocal interface – in order to better highlight the specific issues related to the combination of these modalities and how they have been dealt with through the support of multimodal TERESA.

4.2 The software architecture

From the architectural point of view, in multimodal TERESA we have modules associated with handling each abstraction level managed within the framework (task level, abstract interface level, concrete interface level) and classes devoted to managing the different transformations implemented among such abstraction levels and also to get the final code.

Therefore, apart from the module managing the task level (which is handled by a specific package of classes) and the module for transforming the task-based information into an abstract user interface specification, the abstract/concrete levels of a user interface are controlled by the `EditorFrame`, the module defining the overall structure and behaviour of the visual editor allowing the designer to manage the user interfaces at the abstract/concrete levels. This class calls additional modules delegated to handle the various panels constituting the editor.

For the abstract level, we have a module for handling the list of presentations composing a user interface (`PresentationsListPanel`); another class handling each (abstract) presentation in terms of the nested expressions of composition operators and interactors (`PresentationTreePanel`); another class managing how the connections are specified within each presentation (`ConnectionsPanel`). The `PresentationTreePanel` class, uses the `UIObjectPanel` class for managing the details about the different (abstract) interactors and operators within the presentation. Therefore, `UIObjectPanel` is a general class that handles the information about the various objects of the user interface in a general and platform-independent way. However, depending on the current platform at hand and the object of the abstract user interface currently selected within the tool, the behaviour of the `UIObjectPanel` is dynamically refined into the behaviour of platform-dependent classes, which specify the parameters of the various user interface objects at the concrete level. In addition, the `PresentationTreePanel` also manages the information included in the interaction model, and associated with each presentation.

As for the transformation from an abstract specification of the user interface to the corresponding concrete description, there is an additional module for each type of platform handled within the tool. In the same way, the final transformation, allowing the designer to produce the final code from the various user interface concrete specifications, is handled in separated modules, one for each final language (e.g., XHTML, VoiceXML, X+V, XHTML+SVG, Xlets, etc.) considered within the tool.

5 Platforms supported

5.1 The concrete UI for the desktop platform

In the new tool, we have also improved and made more flexible the possibilities for designing and developing user interfaces for the platforms already supported. For example, we have improved the Concrete User Interface for graphical desktop platform in order to address some issues that were not previously considered in the early versions of TERESA. One of the additions to the concrete user interface for the desktop platform was the introduction of an element having type *interactive description*, as you can see from the following XML snippet:

```
<!ELEMENT interaction (selection | editing | control | interactive_description)>
<!ELEMENT interactive_description (textual | text_link | image_link | button | mailto)+>
```

With ‘interactive_description’, we mean a type of interactive element (beyond selection, editing and control), that can in turn assume different choices, including the possibility of handling mail structures (mailto element). The ‘mailto’ element was also added as a further refinement option for the concrete ‘activator’ object, as you can see from the following snippet:

```
<!ELEMENT activator (reset_button | button_and_script | mailto)>
<!ELEMENT mailto EMPTY>
<!ATTLIST mailto
address CDATA #REQUIRED>
```

Moreover, at the concrete level we included the possibility of describing tables, and their typical components, as you can see from the following definitions:

```
<!ELEMENT table (head?, body, foot?)>
<!ATTLIST table
summary CDATA #IMPLIED
label CDATA #IMPLIED
border NMTOKEN #IMPLIED
colour CDATA #IMPLIED
align (%font_align;) #IMPLIED
height CDATA #IMPLIED
width CDATA #IMPLIED>

<!ELEMENT head (row)>
<!ELEMENT foot (row)>
<!ELEMENT body (row+)>
<!ELEMENT row (table_cell+)>
<!ATTLIST row
colour CDATA #IMPLIED
align (%font_align;) #IMPLIED>

<ELEMENT table_cell (textual | image)>
<!ATTLIST table_cell
vertical_align (%font_align;) #IMPLIED
horizontal_align (%font_align;) #IMPLIED
colour CDATA #IMPLIED >
```

It is worth noting that, for instance, the *colour* attribute was declared as CDATA because it can be expressed in different forms: either using symbolic names or using RGB values. The same holds for *height* and *width* attributes (which can be defined either through number of pixels or through percentage values). Moreover, each cell of the table (*table_cell* element) might contain only specific types of elements (either a textual element or an image). We decided to limit the kinds of elements that can appear as elements of a table, in order to avoid situations such as nested tables.

5.2 Interactive vectorial graphics platform

In order to support interactive vectorial graphics, we had to perform some modifications at the different interface abstraction levels. For instance, *at the abstract level*, in order to enhance the interaction capability within each presentation, an interaction model can be optionally associated with every abstract presentation (in Figure 4, we represented it using a single graphic placeholder without providing further detail in order not to clutter the image). The (optional) specification of an interaction model for each presentation was a key addition for enabling TERESA to support dynamic behaviour even within each presentation. This allows designers to specify objects within a presentation that can be

dynamically created, deleted and modified. It is worth pointing out that, although this addition is especially relevant for the interactive graphics platform (we will provide an example of its application to this platform in Section 6.1), the specification of an appropriate interaction model can also be usefully exploited for adding further interactive capabilities to other platforms (for instance, for adding an interactive behaviour within a web page).

Figure 4 The main structure of an abstract user interface in TERESA (for colours, see online version)

An interaction model is defined in terms of a number of different possible states of the presentation and the set of transitions between these states. Each transition is associated with an initial state, an interactor (triggering the transition), an abstract event and a target state. Because at this stage interaction models are modality-independent, the transitions are defined by means of modality-independent events: selection (choice between different options), editing (modifying objects in the user interface), activation (triggering the execution of a command). Depending on the specific abstract event, various kinds of abstract consequences can be specified accordingly: change state (the effect is moving to a different state), modify interactor (the abstract consequence

of a certain event occurring on a specific interactor results in modifying the same interactor), modify another interactor, conditional consequence (choose between two different abstract consequences), set variable (updating a variable), generic function (define a generic function that should be triggered when the event occurs). Lastly, an empty abstract consequence also means that the related event does not cause any consequence. Of course, at the concrete level, we have to take into account the concrete refinement of the interaction models that at the abstract level are associated with the different abstract presentations. To this aim, at the concrete level, while the state set of the interaction models remains unchanged, the different events will be specified in a platform-dependent manner. For instance, an abstract selection event can be implemented, at the concrete level, by an ‘on click’ event on the current interactor in a desktop environment. Designers are also offered the possibility to associate dynamic behaviours with the concrete transitions. Such behaviours are defined by specifying the functions that will be activated by the concrete events. In this way, designers are provided with some control over run-time presentation behaviours.

Moreover, *at the abstract level*, in order to deal with direct manipulation techniques and also to include structured vectorial graphics in the user interfaces generated by the tool, we decided to substitute the former *object_edit* interactor with a new type of interactor, *the structured object*. A structured object offers the designer two different points of view: from one point of view, it can be seen as a single abstract object that can be uniformly manipulated (*e.g.*, rotated, scaled, transformed). From another viewpoint, a structure within such object can be defined through a number of basic components (basic vectorial graphical elements) combined through the same composition operators we use for composing different interactors (namely: grouping, hierarchy, ordering, relation). Of course, depending on whether such operators are used either to compose *different* interactors or to compose the vectorial graphic elements belonging to the *same* structured object, they will be rendered in a different way because techniques used for rendering operators used to combine vectorial graphical elements belonging to the same object should highlight a stronger relationship, as they not only share some relationships with each other (as it happens in the case of different interactors), but they also belong to the same object. This is rendered by providing additional interactive capability to such structured objects, and, more importantly, such capabilities will be provided so that they can be applied to the *whole* object (in order to emphasise the fact that we are dealing with a single object). For instance, in case of a grouping among different elementary objects, the designer will be able to decide if a particular degree of interactive zooming might be applied: in this case the zooming will be applied to *all* the grouped elements, because they, as a whole, constitute a single object.

5.3 Vocal and graphical multimodal platform

The goal of multimodal TERESA is to provide an authoring environment for flexible development of multimodal interfaces in multidevice environments. Our first multimodal target platform provides for the composition of graphical and vocal interactions. There are many ways to compose such modalities: we have considered four well-known properties (CARE: Complementarity, Assignment, Redundancy and Equivalence) (Coutaz *et al.*, 1995) at various granularity levels and applied such properties in the following manner:

- 1 Complementarity – the considered part of the interface is partly supported by one modality and partly by another one.
- 2 Assignment – the considered part of the interface is supported by one assigned modality.
- 3 Redundancy – the considered part of the interface is supported by both modalities.
- 4 Equivalence – the considered part of the interface is supported by either one modality or another.

Since we want to provide a flexible environment, we support the possibility of applying such properties in the implementation of the various aspects characterising our logical descriptions: the composition operators and the elementary elements. In addition, in order to have the possibility of controlling multimodality at a finer level, the interface elements are structured into three stages:

- 1 Prompt – represents the interface output indicating that it is ready to receive an input.
- 2 Input – represents how the user can actually provide the input.
- 3 Feedback – represents the response of the system after the user input.

Thus, our environment allows the designer to decide what multimodal support to provide for each of the different stages. How such properties will be applied to such elements depends on the modalities and platforms considered. In the case of a desktop system able to support both graphical and vocal modalities, we have to consider that the graphical resources are rich, and thus we have the composition operators supported graphically. The interface elements are structured in such a way that the prompt is graphical, input can be either graphical or vocal, and feedback is in both modalities. In the multimodal PDA, in which the graphical resources are less rich, we have the composition operators supported both graphically and vocally, the interface elements are supported in such a way that the prompt is vocal, the input either graphical or vocal, and feedback in both modalities.

Currently the tool generates multimodal implementation in X+V. This is a W3C standard already supported by freely available browsers such as Opera. Thus, it is possible for all users to access the graphical and vocal interfaces generated. In addition, the support for other implementation languages can be introduced with limited effort, as this would require simply modifying the transformation from the concrete interface description to the target implementation language.

The generated X+V files are divided into three parts. The heading indicates the XML version, the DOCTYPE and the DTD of the language. Then, the tag `<html>` is open to indicate the modules used and contains the head and body. The second part is the head, which includes all the vocal functions and defines the page title and indicates the CSS files to use. The vocal functions are contained in the tag `<vxml:form>`, which contains all the vocal constructs corresponding to the elements composing the concrete interface. The third part is the body, which contains all the graphical HTML constructs corresponding to the elements in the concrete language. In addition, it contains a reference, in the form of event handler, to the tag `<vxml:form>` that manages the vocal part. In the generation of the vocal part our authoring environment is also able to generate the grammar indicating the various possible combinations of vocal input that the application can accept for the

vocal interactions. In particular, at the concrete interface level we allow the designer to edit the grammar for the vocal part at a high level. This representation is then translated into a grammar specification of the implementation language (such as X+V).

5.4 Digital TV platform

As we already stated, we also wish to offer support to devices, such as the digital TV, that are not traditionally connected with the office environment. In the case of the digital TV, the objective was to understand the kind of issues that the consideration of such a platform would have raised in comparison with traditional desktop systems. Indeed, this platform is similar to a graphical desktop, but we have to take into account that in this case users have no mouse or keyword but just a TV controller to interact with it.

In order to guarantee the best readability of the text, we decided to use a specific kind of font – *Tiresias* – which is highly suitable for being visualised on TV displays. In addition, quite high font dimensions were selected (range between 24–36pt), avoiding the smaller ones, which do not guarantee sufficient readability. Moreover, the elements that were connected to the use of Javascript code were deleted, owing to the unavailability of an adequate interpreter.

As for the generation of the user interface implementation, it involves the generation of a file in a Java version for digital TVs representing an *xlet*, which is a particular application that is immediately compiled and can be interpreted and executed by the interactive TV decoders. Xlets bear a strong resemblance with common Java applets, with the difference that, instead of the web browser (which executes the applets), there is the MHP¹ layer of the digital receiver (Set-Top-Box) which interprets them. Once we have the xlet, in real settings (which usually means that a digital phone line is available), the file generated with TERESA tool (the xlet) is supposed to be downloaded on the Set-Top-Box. If such a line is not available, an emulator can be used to execute the interactive Xlet. For our examples, we used the XletView emulator;² we will show an example of its use in Section 6.3. As we said before, the Xlets are similar to the applets. However, there is a big difference regarding the user interface part: Xlets do not include the AWT package, which contains several widgets, such as radio button, check box, button, *etc.* Thus, we have also developed a library that provides such techniques and simplify the target for the user interface generation in the digital TV platform. In the library there is a class that provides the methods to include the widgets in the application without having to specify all the details every time. Then, there is a class for each interactor implementing its appearance and behaviour. The implementation of the appearance is not trivial because the basic Xlets just provide primitives for drawing rectangles and showing images. The semantics of the interactors' implementation is managed through specific event handlers that allow them to update the state and then their appearance accordingly.

5.5 Gesture and graphical multimodal mobile platform

The important role of gestures must be borne in mind in the design of multimodal interfaces. While some work has been dedicated to sketching as a tool to ease user interface design (Landay and Myers, 2001), little attention has been paid to the model-based generation of gesture user interfaces. Gestures can potentially be used in

navigation and in providing control commands. So, a user interface supporting gesture interaction must combine at least the graphical and gesture modalities. In addition, vocal or tactile modalities can be included in mobile devices. To illustrate it, let us consider the basic elements of an interaction: prompt, input and feedback. A complete interaction with a mobile device supporting gesture can include several modalities: graphical for the prompt, gestures for providing input, and vocal, tactile and/or graphical for feedback.

In addition to fixed naming of gestures, there must be support for defining customised gestures. Since inputting (control, activate, select and navigate) is the primary form of interaction for gestures, and it is difficult to prompt and give feedback by using gestures, various types of interaction elements must be carefully combined with other modalities to ensure a functional final user interface for a target device.

The XML concrete gesture interface language is a refinement of the abstract TERESA language (as it happens for all platforms). In this case it also provides support for tilt-gestures, which can be associated with the interaction types supported.

The concrete gesture interface language defines an interface using default settings and a number of presentations, each of which consists of definitions of the relevant interactors ('interaction' or 'only output') as well as the composition operators.

In the following, the CUI declarations are presented. For all presentations, the gesture settings define which gesture sets are used:

```
<!ELEMENT default_settings (graphic_settings, gesture_settings,
operators_settings, interactors_settings)>
<!ELEMENT gesture_settings (change_focus*, gesture_sets*)>
<!ELEMENT gesture_sets (tilt*)>
```

The main function of gestures in mobile user interfaces is simply to navigate between interface elements. For gesture UI it is crucially important to highlight the current focus of interaction. The method for changing the focus is defined in the default settings. In our example changing the focus is done by using basic tilt commands:

```
<!ATTLIST change_focus
next (%tilt_act;) #REQUIRED
previous (%tilt_act;) #REQUIRED>
```

Showing the focus graphically on a target platform can be done by modifying graphical elements, such as highlighting the text, outlining the box, changing the colour, *etc.* In the tool, the gesture actions can be associated with interactors, and they are defined by default with the following tilt-based sets:

```
<!ENTITY % tilt_act "null | tilt_left | tilt_right | tilt_forward | tilt_backward">
```

In multimodal UI, gestures can be potentially used only for input. In particular, they are suitable for control and selection. The abstract interactors that were judged relevant to be used for providing input with gestures are 'control' and 'selection'. As you can see from Figure 4, the *control* interactor includes navigator and activator elements, while the *selection* interactor includes single and multiple selection elements. Such types of interactions also need to operate in parallel with graphical elements such as buttons or various types of lists. Graphically, interactors can be implemented using many objects, such as links, various types of buttons, menus and boxes. Input interactors for gestures are defined as extensions to existing graphical ones.


```
<!ELEMENT activator (button, gesture*)>
<!ELEMENT navigator (button, gesture*)>
<!ELEMENT single ((radio_button | list_box | drop_down list) gesture*)>
<!ELEMENT multiple ((check_box | list_box), gesture*)>
```

where gesture element is defined with a given gesture set with possible gesture entities.

If we analyse the possible use of the modalities through the CARE properties in the gesture and graphical mobile platform, it is reasonable to output information only graphically, so the property for outputting data is graphical assignment, as well as for providing prompt and feedback. A user should be able to enter input using both graphical and gestural modalities. Thus, the property for inputting information is equivalence, while graphical assignment can be another option. In the case of composition operators (grouping, hierarchy, ordering and relation), the only relevant property is graphical assignment for the same reason as in the ‘only output’ interactor (the gestural modality cannot be used to output information).

6 Example applications

In this section, we provide some example application for each new target platform addressed by multimodal TERESA.

6.1 Examples with interactive vectorial graphics platform

In this first example we show an interactive slide show application in which the user is allowed to select a specific image, then optionally interact with it through some actions that are provided within the presentation (*e.g.*, rotating the image, or moving it or zooming in/out) and then, when the best view of the image has been selected, insert a textual comment within a textfield devoted to include a comment on the picture, and such comment will also appear as a text into the bottom panel.

In this case the related presentation is constituted by a ‘Text Edit’ interactor for supporting the task of commenting on the photo, and two graphical objects (one for rendering the photo, and the other one for presenting the text) grouped together. It is worth pointing out that the graphical object for presenting the text was rendered through a vectorial graphic-based object (and not a bare textual element) in order to support interactivity – in this case position, orientation and size can be obtained through the user’s direct manipulation. Of course, through multimodal TERESA, the designer can specify further concrete details about how to implement the various abstract objects. For instance, in this case, the (yellow) stroke parameter for the text was appropriately translated for rendering the text (as shown in Figure 5).

In Figure 5, the corresponding user interface is displayed. In Figure 5(b) you can see the resulting page after user interaction on the user interface visualised in Figure 5(a), which means that the user rotated the currently displayed picture, zoomed in on it, and moved it so as to have the interesting part (the foreground sledge) in the centre of the presentation. After such actions, the user added a comment (by filling in the textfield) and such a comment was also rendered as a graphical textual object that the user in turn rotated, zoomed in on and moved onto the most appropriate place of the presentation

(near the concerned subject in the photo). It is worth pointing out that, in order to directly manipulate any interactive graphical object included in the presentation (e.g., the picture about the winter holiday, or the related textual comment), the user had to click on the textual element identifying the action (e.g., the text ‘Move’, or ‘Rotate’, etc.). This interactive behaviour is the result of concretely translating the behaviour described within the interaction model associated with the presentation considered in the example, which is shown in Figure 6. This is performed by including in the SVG snippets referring to the graphical objects, appropriate scripts implementing the dynamic behaviour of such objects.

Figure 5 The initial presentation of a page created with TERESA (a – left) and the same page after user interaction (b – right)

As you can see, in correspondence with the abstract event ‘on select’ occurring on the textual interactor, an abstract consequence of type ‘change state’ is associated. At the concrete level the abstract ‘on select’ event has been translated into the concrete ‘on click’ event that will trigger a change of state: from the current state s_2 , to the target state s_5 in which the user will be enabled to actually move the graphical objects in the presentation (the photo, in our case).

In the next example, we suppose that the director of a company wishes to have an overview of the income gained by the company through the last three years, in order to follow the evolution over a single year and also possibly compare figures over the different years. Therefore, at the task level we specify not only having an overview and comparing figures for the different years, but also that the comparison will manipulate three numerical objects, as each of them stands for data related to income gained over (the months of) a different year. All the data we specify at the task level are useful to derive, at the abstract level, a suitable interactor supporting the specified activity, in this case the *ordered structured object* (see Figure 7, bottom-right panel).

Figure 6 The TERESA window for specifying the interaction model at both abstract and concrete levels

Interaction Model Editing

States

- 1 - state: s0
- 2 - state: s1
- 3 - state: s2
- 4 - state: s3
- 5 - state: s4
- 6 - state: s5
- 7 - state: s6

Events

#	Interactor Id	Abstract Event	Abstract Consequence	Consequence
0	Text_1_0	onselect	change_state	change_state(s1)
1	Text_1_1	onselect	change_state	change_state(s3)
2	Text_1_2	onselect	change_state	change_state(s4)
3	Text_1_3	onselect	change_state	change_state(s5)
4	Text_1_4	onselect	change_state	change_state(s6)
5	Text_Edit_1_0	onedit	modify_other	set attribute (Object_...
6	Single_Selection_1_0	onselect	if_then_else	if (value==val2) then ...

State Events / State Initializations / Functions

Concrete Attributes

Interactor Id: Text_1_3

Interactor Type: Text

Abstract Event: onselect

Consequence: change_state

Implementation Techniques

Associated Event: onclick

Target state: s5

Apply

When moving to the next, concrete level, such abstract information has to be further refined. For instance, as you can see from Figure 7, one of the possible concrete representations for the concerned ordered structured object is a line chart (other possible representations might be barchart, piechart, *etc.*) and the designer has specified that the vertical axis will show the 'Kiloeuro' text and the chart will be zoomable up to 40%.

In addition, the fact that the line chart has to display three different lists of numerical values was already available at the abstract level but, at this step, the designer further makes concrete such a specification by including the real values for the three different years with the opportune ordered lists of value (it can be seen from the superimposed window on the left part of Figure 7). The last phase of the process deals with the actual generation of the code of the final user interface. The result of such phase for the desktop system is displayed in Figure 8, where you can see the final layout of the line chart considered in the example.

Figure 7 The TERESA window for specifying the interaction model **Figure 8** The final desktop presentation of the ordered structured object (for colours, see online version)

6.2 Example with vocal and graphical platforms

A number of applications have been developed to test the multimodal features of the tool. One concerned a cinema application. When users access the home page they can first select a town, then they can choose the movie that they want to see; lastly, they can make a reservation indicating the preferred seat. A picture of the tool while authoring such an application was already provided in Figure 3 (Section 4), which shows the logical description of the interface that can be obtained either by editing it or as a result of an automatic transformation from the task model description. In the case of automatic transformation from the task model, then the designer still has to edit the values of the concrete attributes in order to make the resulting user interface more appealing.

In Figure 3, we can see that the top-left part shows the list of the defined presentations. The currently selected one concerns the movie 'Madagascar'.

The text and the description elements are obtained through the graphical modality. The description consists of text with the support of some images. The two navigator elements allow the user to move to the reservation part or to the home page. The input for selecting the next page to access can be provided equivalently either through the vocal or graphical modality. The prompt is given by the label of the two corresponding buttons. In addition, for highlighting such possibilities, a redundant vocal prompt is given ('Say book to buy seats for the movie, or back to return to the home page').

Figure 9 shows how this type of presentation is generated for a mobile device. In this case, while the logical structure of the page is still the same, there are changes on how the multimodality is supported. The vocal modality is much better exploited because of the limited graphical resources. Thus, some information is provided only vocally, some are provided both vocally and graphically and some are provided by exploiting the two modalities in a complementary way.

Figure 9 The resulting multimodal interface

6.3 Example with digital TV platform

For our example of supporting digital TV platform, we used the Carrara Marble Museum application. We do not show the user interface of TERESA for setting the different configuration parameters in the concrete user interface, because it does not present any particular difference with respect to, *e.g.*, the logical description for the desktop system.

Figure 10 shows the user interface for the introductory page of the Marble Museum, as it is rendered on the emulator of the digital TV. As you can see, it is possible to use different buttons of the TV controller for moving within the user interface and select the different sections (General Information, Access to Artworks, Booking Ticket). If we select the Booking Ticket section, the user interface that will be presented to the user is shown in Figure 11.

Figure 10 The digital TV-based museum page

Figure 11 Booking a ticket

As you can see from Figure 11, there is a part in which it is possible to fill the information requested from the user and proceed with reservation, and such an editing activity is performed by using a virtual keyboard.

6.4 Example with tilt and graphical mobile platform

This section illustrates an application of model-based design multidevice application for a tilt and graphical user interface in a mobile device and for graphical user interface in a desktop. For generating UI for a desktop application the design follows the transformation defined in the initial TERESA (Mori *et al.*, 2004).

The type of application that we have considered for the mobile device is form-based, meaning that the interactive part of the user interface mainly consists of interactive forms. The implementation for the selected target platform is in Microsoft C# (supported by Visual Studio 2005), we assume that there is a project for containing libraries or an interface to a software producing gesture events (TiltManager – recognition software).

In Microsoft Visual Studio 2005 a user interface is (mainly) defined through the Windows Forms, which, in C#, are defined as <name>.cs and <name>.designer.cs-files. The first type of file contains source for defining functionality of the form, together with other sources files, while the latter type of file defines graphical layout of a window (Form). The <name>.designer.cs-files can be created traditionally by typing or automatically when designing layout by drag-and-drop in the Designer tool of the Interface Development Environment. In generating graphical and gesture user interfaces for mobile devices, our environment creates an implementation with two types of files: <name>.designer.cs files and <name>.cs files. None of the previous approaches to tilt-based interaction (Eslambolchilar and Murray-Smith, 2004; Hinckley *et al.*, 2000; Mäntyjärvi *et al.*, 2005) has exploited logical descriptions to support their design.

The final gesture-based application implementation for a mobile device includes (in addition to files of the executable application) the description of the adapted layout of the graphical part through .designer.cs file with the corresponding pieces of code required by the application and the event-handling codes necessary for defining interaction with the TiltManager software. A specific code defines the highlighting of focus and gesture commands supported by the gesture recognition software on a target platform. These parts are generated according to the abstract and concrete user interface specifications and information about the target platform.

In our prototype supporting tilt-based gestures, the gesture events are TiltLeft, Tilt-Right, TiltBackward and TiltForward. Gesture events are interpreted as state transitions between interactor elements. In the implementation of the final user interface, these commands are set as alternative choices for activating defined controllers graphically (when the modalities are equivalent). Setting the focus graphically according to tilt-gestures is carried out by generating focus-related methods and properties to the user interface.

In the example we consider a mobile terminal supporting graphical and gesture modalities, and appearance and functionality of the application are adapted accordingly. The user interfaces for desktop and for gesture mobile device are shown in Figures 12 and 13, respectively. The accelerometer able to sense gestures is the small box by Ecertech at the bottom of the PDA (see Figure 13).

Figure 12 The user interface for a graphical desktop

The screenshot shows a web browser window titled "Marble Museum - Microsoft Internet Explorer". The page has a light blue background with a subtle pattern. At the top, there are three small images: a museum interior, a museum exterior, and a marble statue. Below the images, the text reads: "Dear Customer, In order to join our mailing list some basic information from You is required. The information is used for tailoring our services to Your needs." The form fields are as follows: "First Name" with the value "Giovanni", "Last Name" with the value "Verdure", "Country" with a dropdown menu showing "Austria", "Age" with radio buttons for "10-20", "21-30", "31-40", "41-50", "51-60", and "61+", "Sex" with radio buttons for "Male" and "Female", and "Education" with a dropdown menu showing "No". At the bottom, there is a "Thank you for getting in touch with us." message and "Submit" and "Cancel" buttons.

Figure 13 The user interface for a graphical+gestural mobile device

The screenshot shows a handheld PDA device with a screen displaying the "Marble Museum Form". The screen has a light blue background. The text on the screen reads: "Registration Form", "Dear Customer, In order to join our mailing list some basic information from You is required. The information is used for tailoring our services according to Your needs". The form fields are: "First Name:" with the value "Giovanni", "Lastname:" with the value "Verdure", "Country:" with a dropdown menu showing "Austria", "Age:" with a dropdown menu, "Sex:" with radio buttons for "Female" and "Male", and "Education:" with a dropdown menu. At the bottom, there are "Ok" and "Cancel" buttons. The device is a black handheld PDA with a small screen and a few buttons visible.

Since the mobile device interface supports graphical and gesture modalities, *i.e.*, the user interface is adapted according to the features of the target device. An analysis of the application reveals that input interactors and images are provided in the same presentation on a desktop registration form. As it can be seen from Figure 12, the graphical interactors supported in parallel are text edit for entering name (character input field), single selection for selecting country and education (drop-down menu), age and sex (radio button), activator for cancelling and navigation for submitting a filled form (buttons).

Instead, the mobile device user interface displays the graphical presentation of the input interactors described above in a layout suited for its graphical properties. Images are not shown on a mobile device and labels of the interactors are shortened. Single selection regarding Age is changed to a drop-down list, which is suitable for small screen. Single selection concerning Sex is changed from horizontal alignment to vertical. Gesture actions to interactors in our example application can be performed by tilting right. Single selection (selecting country from a list, selecting age from a list, selecting sex) is carried out by tilting backward and forward to find suitable object, and making selection by tilting right.

In implementing the input part, the equivalence property is applied, meaning that either modality, gesture or graphical (stylus), can be used for inputting. For feedback and prompting, the graphical modality is assigned. In the current version no redundant or complementary use of the two modalities (graphical and gestural) takes place. Compared to traditional graphical user interfaces on a mobile device, the gesture modality in this case provides an alternative way of entering input in the application.

7 Conclusions and future work

While model-based approaches to user interface design have been introduced a long time ago, little work has been dedicated to the support of a wide variety of platforms, which can vary also in terms of interaction modalities.

In this paper we have shown how multimodal TERESA is able to support many platforms (interactive vectorial graphics, vocal and graphical, digital TV, gestural and graphical, *etc.*). This has implied a revision of the original abstract description and the introduction of concrete descriptions for each new platform considered.

In the paper we have introduced the support for such platforms and corresponding example applications. To obtain such results in addition to the necessary XML-based descriptions, we have implemented the corresponding transformations among the various conceptual levels involved (task, abstract interface, concrete interface, implementation) and included them in our authoring environment. Additional work is currently going on about the possibility for the tool to reconstruct the design of legacy systems implemented in various languages through a reverse engineering process, so as to deliver more abstract specifications that could be used, for example, to support a semantic redesign of a product, or even a more general round-trip engineering approach.

Future work will be dedicated to making the support of the transformations generating implementations for the various target platforms more easily extendable, modifiable and documented so that adding new transformations or modifying existing ones can be done without modifying the authoring environment implementation.

References

- Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S. and Shuster, J. (1999) 'UIML: an appliance-independent XML user interface language', *Proceedings of the 8th WWW Conference*.
- Ball, T., Colby, C., Danielsen, P., Jagadeesan, L.J., Jagadeesan, R., Läufer, K., Mataga, P. and Rehor, K. (2000) 'Sisl: Several Interfaces, Single Logic', *International Journal of Speech Technology*, Kluwer Academic Publishers, Vol. 3, No. 2, pp.91–106.
- Bastide, R., Navarre, D., Palanque, P., Schyn, A. and Dragicevic, P. (2004) 'A model-based approach for real-time embedded multimodal systems in military aircrafts', *Proceedings of ICMI 2004*, ACM Press, pp.243–250.
- Berti, S., Correani, F., Paternò, F. and Santoro, C. (2004) 'The TERESA XML language for the description of interactive systems at multiple abstraction levels', *Proceedings of Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages*, May, pp.103–110.
- Berti, S. and Paternò, F. (2003) 'Model-based design of speech interfaces', *Proceedings of DSV-IS 2003*, Springer Verlag, LNCS 2844, pp.231–244.
- Bouchet, J., Nigay, L. and Ganille, T. (2004) 'ICARE software components for rapidly developing multimodal interfaces', *Proceedings of ICMI 2004*, ACM Press, pp.251–258.
- Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. and Vanderdonckt, J. (2003) 'A unifying reference framework for multi-target user interfaces', *Interacting with Computers*, June, Vol. 15, No. 3, pp.289–308.
- Chesta, C., Paternò, F. and Santoro, C. (2004) 'Methods and tools for designing and developing usable multi-platform interactive applications', *PsychNology Journal*, Vol. 2, No. 1, pp.123–139.
- Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J. and Young, R. (1995) 'Four easy pieces for assessing the usability of multimodal interaction: the CARE properties', *Proceedings of INTERACT*, pp.115–120.
- Eslambolchilar, P. and Murray-Smith, R. (2004) 'Tilt-based automatic zooming and scaling in mobile devices – a state-space implementation', *Proceedings of MobileHCI 2004*, LNCS 3160, Glasgow, UK: Springer-Verlag, 13–16 September, pp.120–131.
- Hinckley, K., Pierce, J., Sinclair, M. and Horvitz, E. (2000) 'Sensing techniques for mobile interaction', *ACM UIST 2000 Symposium on User Interface Software & Technology*, CHI Letters, Vol. 2, No. 2, pp.91–100.
- Landay, J. and Myers, B. (2001) 'Sketching interfaces: toward more human interface design', *IEEE Computer*, March, Vol. 34, No. 3, pp.56–64.
- Mäntyjärvi, J., Kallio, S., Korpipää, P., Kela, J. and Plomp, J. (2005) 'Gesture interaction for small handheld devices to support multimedia applications', *Journal of Mobile Multimedia*, Rinton Press, Vol. 1, No. 2, pp.92–112.
- Mori, G., Paternò, F. and Santoro, C. (2004) 'Design and development of multi-device user interfaces through multiple logical descriptions', *IEEE Transactions on Software Engineering*, IEEE Press, August, Vol. 30, No. 8, pp.507–520.
- Mullet, K. and Sano, D. (1995) *Designing Visual Interfaces*, Prentice Hall.
- Obrenovic, Z., Starcevic, D. and Selic, B. (2004) 'A model-driven approach to content repurposing', *IEEE Multimedia*, January–March, pp.62–71.
- Paternò, F. (1999) 'Model-based design and evaluation of interactive application', Springer Verlag, ISBN 1-85233-155-160.
- Puerta, A. and Eisenstein, V. (2001) 'XIML: a common representation for interaction data', *Proceedings of ACM IUI 2001*, pp.214–215.

- Stanciulescu, A., Limbourg, Q., Vanderdonckt, J., Michotte, B. and Montero, F. (2005) 'A transformational approach for multimodal web user interfaces based on USIXML', *Proceedings of ICMI 2005*, ACM Press, pp.259–266.
- Sukaviriya, P. and Foley, J. (1993) 'Supporting adaptive interfaces in a knowledge-based user interface environment', *Proceedings of Intelligent User Interfaces Conference*, ACM Press, pp.107–113.
- Szekely, P. (1996) 'Retrospective and challenges for model-based interface development', *2nd International Workshop on Computer-Aided Design of User Interfaces*, Namur: Namur University Press.

Notes

- 1 <http://www.mhp.org/>
- 2 <http://xletview.sourceforge.net>