

DESIGNING MULTI-DEVICE INTERACTIVE SERVICES THROUGH MULTIPLE ABSTRACTION LEVELS

SILVIA BERTI, GIULIO MORI, FABIO PATERNO', CARMEN SANTORO

ISTI-CNR, Pisa

{silvia.berti, giulio.mori, fabio.paterno, carmen.santoro}@isti.cnr.it

The purpose of this paper is to discuss the features of our TERESA environment and show how it can support design and development of multi-device interactive services. This is a tool for model-based design of multi-device interfaces. It considers three levels of abstractions (task model, abstract user interface and concrete user interface). For each of them a specific language has been defined and used. The task and the abstract user interface levels are described by platform-independent languages, which allow designers to focus on conceptual aspects and to avoid dealing with a plethora of low level details.

1 Introduction

With the advent of the wireless Internet and the rapidly expanding market of smart devices, designing interactive applications supporting multiple platforms has become a difficult issue. The main problem is that many assumptions that have been held up to now about classical stationary desktop systems are being challenged when moving towards nomadic applications, which are applications that can be accessed through multiple devices from different locations. Consequently, one fundamental issue is how to support software designers and developers in building such applications. In particular, there is a need for novel methods and tools able to support development of interactive software systems that adapt to different targets while implementing usability design criteria.

Model-based approaches [7] could represent a feasible solution for addressing such issues: the basic idea is to identify useful abstractions highlighting the main aspects that should be considered when designing effective interactive applications. Our approach extends previous work in the model-based design area in order to support development of nomadic applications through logical descriptions and associated transformations.

By 'platform' we mean a class of systems that share the same characteristics in terms of interaction resources. Examples of platforms are the graphical desktop, PDAs, mobile phones and vocal systems. Their range varies from small devices such as interactive watches to very large flat displays. While designers should be aware of the potential platforms (not devices) early on in the design process in order to identify the tasks suitable for each of them, our method allows them to avoid dealing with a plethora of low-level details because the last transformation (from concrete to implementation) is automatic. In addition, the same languages are used to describe tasks and abstract interfaces for all platforms; only the language for describing

concrete user interfaces is to some extent platform-dependent (it is actually a platform-dependent refinement of the abstract user interface description).

To support this approach, the TERESA (Transformation Environment for inteRactivE Systems representAtions) tool [5] has been designed and developed providing general solutions that can be tailored to specific cases. This tool supports transformations in a top-down manner, providing the possibility of obtaining interfaces for different types of devices from logical descriptions. It differs from other approaches such as UIML [1], which mainly consider low-level models. XIIML [9] has similar goals but there is no publicly available tool supporting it.

Some usability criteria are incorporated into the tool transformations from task to user interface. This means that the tool is able to provide suggestions for selecting the most appropriate interaction techniques and ways to compose them. Such transformations guarantee a consistent design because the same design criteria are applied in similar situations. In addition, most of the functionality of the CTTE task modelling tool [4] have now been integrated into TERESA, so that designers can use just one tool for developing and analysing task models and for generating different interfaces that adapt to various interaction platforms.

2 Logical Design of Multi-device Interfaces

Logical design of multi-device interfaces can be supported by a number of steps that allow designers to start with an overall envisioned task model of a nomadic application and then derive concrete and effective user interfaces for multiple devices:

- *High-level task modelling of a multi-context application.* In this phase designers develop a single model that addresses the possible contexts of use and the various roles involved and also identify all the objects that have to be manipulated to perform tasks and the relationships among such objects. Such models are specified using the ConcurTaskTrees (CTT) notation [7], which also allows designers to indicate the platforms suitable to support each task.
- *Developing the system task model for the different platforms considered.* Here designers have to filter the task model according to the target platform and, if necessary, further refine the task model, depending on the specific device considered, thus, obtaining the system task model for the platform considered.
- *From system task model to abstract user interface.* Here the goal is to obtain an abstract description of the user interface composed of a set of presentations that are identified through an analysis of the task relationships. Each presentation is structured by means of interactors composed of various operators.

- *User interface generation.* In this phase we have the generation of the user interface. This phase is completely platform-dependent and has to consider the specific properties of the target device.

The advantage of this approach is that it is able to address all the possible relations between tasks and platforms. In general, when a multi-platform application is considered, it is important to understand what types of tasks can actually be performed in each available platform. We have identified a number of possibilities:

- *The same task can be performed on multiple platforms in the same manner.* There may be only some changes in attributes of the user interface objects from platform to platform. For example, a login is often performed in almost the same manner through different platforms.
- *Same task on multiple platforms but with different user interface objects.* An example of this case can be a selection task. One platform can support a graphical selection whereas another one can be able to support only a selection among textual links.
- *Same task on multiple platforms but with different domain objects.* This means that during the performance of the same task different sets of domain objects are presented, taking into account the interaction resources available.
- *Same task on multiple platforms but with different task decomposition.* This means that the task is sub-divided differently, with different sets of sub-tasks, depending on the platform.
- *Same task on multiple platforms but with different temporal constraints.* In this case the difference is in the temporal relationships among the subtasks. For example, vocal interfaces tend to serialize interactions that can be performed concurrently on graphical interfaces.
- *Dependencies among tasks performed on different platforms.* An example of this can be found in applications where the users can reserve their flight reservation through a desktop system, and this enables the possibility of getting real-time information regarding the flight through a mobile phone.

3. TERESA

TERESA is intended to provide a complete semi-automatic environment supporting a number of transformations useful for designers to build and analyse their design at different abstraction levels and consequently generate the user interface for various types of platforms.

3.1 Requirements

A number of main requirements have driven the design and development of TERESA:

- *Mixed initiative*; we want a tool able to support different levels of automation ranging from completely automatic solutions to highly interactive solutions where designers can tailor or even radically change the solutions proposed by the tool.
- *Model-based*, the variety of platforms increasingly available can be better handled through some abstractions that allow designers to have a logical view of the activities to support.
- *XML-based*, each abstraction level considered can be described through an XML-based language.
- *Flexible development*, our approach aims to be comprehensive and to support various possibilities, including also when different set of tasks can be performed on different platforms. However, there can be cases where only a part of it needs to be supported and, for example, designers want to start with a logical interface description and not with a task model. In addition, while the original version of the TERESA tool supports only top-down transformations, it can easily be integrated with reverse engineering techniques in order to support the possibility of starting with something concrete, redesign it and obtaining new implementations.
- *Web-oriented*, we decided that Web applications should be our first target. However, the approach can be easily extended to other environments (such as Java applications, Microsoft environments, ...) by just modifying only the last transformation (from concrete interface to final interface).

The TERESA tool offers a number of transformations and provide designers with an integrated environment that allows them to take a logical description and obtain a more refined one taking into account the features of the target platforms and generate interfaces for desktop, mobile phones and vocal devices. This solution allows designers to concentrate on conceptual aspects without having to deal with a plethora of low-levels details. The corresponding final user interfaces are generated automatically.

Currently, various user interface implementation languages are supported (XHTML, XHTML Mobile Profile, VoiceXML) and support for others (such as X+V and SVG) is under development. In case a new implementation language has to be supported just the last transformation from the concrete interface description (the

most refined logical description) to it has to be implemented, without changing the other transformations or the tool architecture.

As Figure 1 shows, the starting point of the design process can vary. It is possible to start with the task model of the nomadic application where designers can specify for each task what platform is suitable to support it and the possible temporal relations among tasks supported by different devices. However, it is also possible to start the development process with the task model or the abstract description of a given platform.

The task and the abstract levels are described by two device independent languages whereas the concrete level is described through platform-dependent refinements of the abstract level in order to be able to describe the platform attributes and specify the values that define their state.

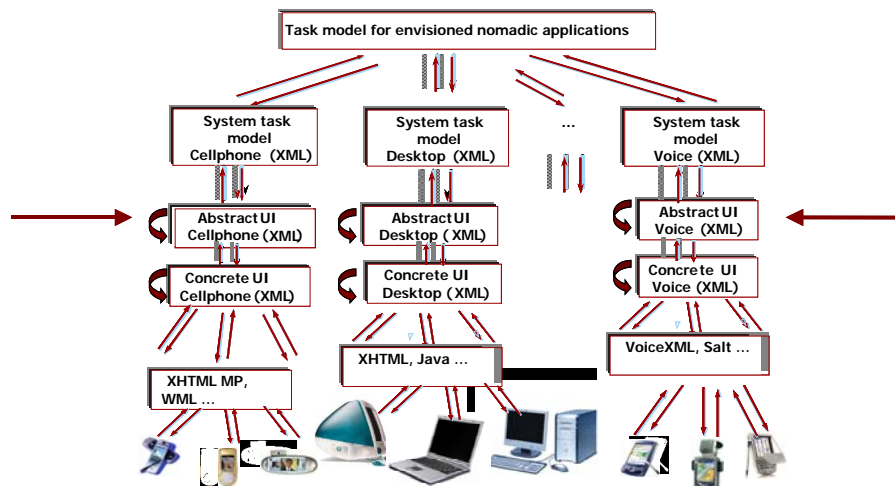


Figure 1 The TERESA Method

The tool is able to keep relations between elements in one abstraction level and the corresponding elements in another level. For instance, using TERESA it is possible to know which interactor corresponds to a certain task, but also access the inverse mapping, since for each interactor the tool is able to automatically identify and highlight the related task, so that designers can immediately spot such a relation. This is particularly useful especially when it comes to specifying the properties of each interactor, as the knowledge of the task it supports is an important indication of its meaning and goal, so it helps designers to position the interactor within the overall application and decide on the most appropriate settings.

The tool also supports semantic platform-dependent redesign. We mean for semantic platform-dependent redesign the possibility of changing the design for an interactive platform in order to adapt to a new one. Semantic redesign means that

this transformation is based on the use of semantic information and not only on the analysis of the low-level implementation [3]. In our case, the semantic information is in the logical descriptions of the user interfaces that capture the possible tasks that users intend to accomplish. The logical descriptions and the transformations defined in the method presented can also be used at run-time to support migratory interfaces: interfaces able to dynamically move from one device to another while preserving interaction continuity and adapting to the features of the new device [2].

3.2 Main Functionality

As we mentioned before, TERESA is a transformation-based tool that supports the design of an interactive application at different abstraction levels and generates the concrete user interface for various types of platforms. The main transformations supported in TERESA are:

- *Presentation task sets and transitions generation.* From the specification of a CTT task model concerning a specific platform, it is possible to obtain the *Presentation Task Sets* (PTSs), sets of tasks which are enabled over the same period of time according to the constraints indicated in the model and *transitions* specifying the conditions allowing moving across PTSs. Such sets, depending on the designer's application of a number of heuristics (general criteria used to merge together two or more PTSs) supported by the tool, can be grouped together so identifying the groups of tasks that should be supported by each user interface presentation.
- *From task model -related information to abstract user interface.* The goal of this phase is mapping the task-based specification of the system onto an interactor-based description of the related abstract user interface. Both the task model and Presentation Task Sets specifications are the input for the transformation generating the associated abstract user interface. The specification of the abstract user interface, in terms of both its static structure (the "presentation" part) and dynamic behaviour (the "dialogue" part), is saved for further analyses and transformations.
- *From abstract user interface to concrete interface for the specific platform.* This transformation starts with the loading of an abstract user interface previously saved and yields the related concrete user interface for the specific media and interaction platform selected. A number of parameters related to the customisation of the concrete user interface are made available to the designer.

- *Automatic UI Generation.* The tool automatically generates the final UI for the target platform. The starting point can be either the single-platform task model, using a number of default configuration settings related to the user interface generation, or the abstract or the concrete user interface.

4. The TERESA Abstract User Interface

An abstract user interface is composed of a number of presentations and connections among them. Each presentation defines a set of presentation and interaction techniques perceivable by the user at a given time. The connections define the dynamic behaviour of the user interface. More precisely, they indicate what interactions trigger a change of presentation and what the next presentation is. They can be associated with conditions in case a specific combination of interactions should trigger the change of presentation.

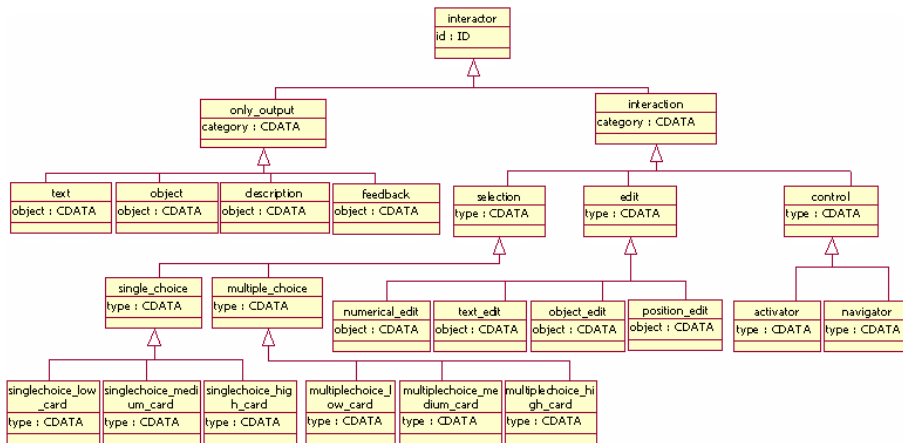


Figure 2 The interactors' hierarchy implemented in TERESA

The structure of the presentation is defined in terms of interactors (abstract descriptions of interaction objects classified depending on their semantics) [8] and their composition operators. In Figure 2 the specification of the various interactors is displayed. As you can see, it is in the shape of a hierarchy, with those at the lower levels inheriting properties of those at the higher levels. The advantage of deriving interactors by successive inheritance is in that it enables exploring the underlying design space of the most suitable interaction objects to support the current task.

It is possible to distinguish between interactors supporting user interaction (*interaction* elements) and those that present results of application processing (*only_output* elements). The interaction elements imply an interaction between the user and the application. There are different types of interaction elements depending

on the type of task supported. We have selection elements (to select between a set of elements), edit (to edit an object), control (to trigger an event within the user interface, which can be useful to activate either a functionality or the transition to a new presentation). Differently, an *only_output* element defines an interactor which implies an action only from the application. There are different types of *only_output* elements (text, object, description, feedback) depending on the type of output the application provides to the user: a textual one, an object, a description, or a feedback about a particular state of the user interface.

The composition 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 [6]. They are:

- *Grouping (G)*: indicates a set of interface elements logically connected to each other;
- *Relation (R)*: highlights a one-to-many relation among some elements, one element has some effects on a set of elements;
- *Ordering (O)*: some kind of ordering among a set of elements can be highlighted;
- *Hierarchy (H)*: different levels of importance can be defined among a set of elements.

5. From Task Models to Abstract User Interfaces

Within this step the CTT task model is translated into an equivalent representation expressed in terms of *Presentation Task Sets* (PTS), sets of tasks enabled over the same period of time according to the constraints indicated in the task model. Each PTS then identifies the group of activities that should be supported by each abstract user interface presentation at the same time. The conditions allowing moving across PTSs are also derived from the task model, and called *transitions*. In order to reduce the number of PTS which, in some cases might be very high, a number of *heuristics* can be applied. The application of the various heuristics is supposed to be triggered by the designer, while TERESA automatically calculates the resulting sets. The motivations for merging together two or more presentation task sets could be reducing the number of PTSs (which can be very high in some cases), or including within the same presentation significant information (as a data exchange is), even when the involved tasks should belong to different PTSs, so that users can better follow the flow of information. Up till now, a number of heuristics has been identified (see bullet list below).

- *Joining when Enabling.* If two (or more) PTSs differ for only one element, and those elements are at the same level connected with an enabling operator, they can be joined together.
- *Single Element Sets.* If a PTS is composed of just one element, it can be included within another superset containing its element.
- *Sharing Most Elements Sets.* If some PTSs share most elements, they can be unified in order not to duplicate information which is already available in another presentation in almost all parts. For example if the common elements all appear at the right of the disabling operator, they can be joined into one PTS.
- *Exchanging Information.* If there is an exchange of information between two tasks, they can be put in the same PTS in order to highlight such data transfer.

Lastly, the tasks included into the various PTSs are translated into suitable groups of interactors supporting the performance of such tasks, and such groups are represented in structured expressions composing the ‘abstract structure’ of each presentation through some composition operators. It is worth pointing out that the interactor composition operators that appear in the abstract user interface are derived by analysing the CTT task model specification., by analysing not only CTT operators, but also other attributes of the tasks (such as frequency). For example, in the case of the Hierarchy operator the application rule strongly depends on the frequency values of the tasks involved. A high level of task frequency is indication that a task is recurrently performed, so it has greater ‘importance’ with respect to other tasks that are less frequently performed: the hierarchy operator is appropriate for conveying this kind of information.

This phase also deals with appropriately translating transitions appearing within the PTS-based description onto appropriate interactors linking the various abstract presentations (we called them *connections*), so describing the dynamic behaviour of the system. In addition, some links with the functional core are identified.

6. From the Abstract User Interface to Its Implementation

One of our main goals in designing TERESA was to provide a flexible environment for designers following a mixed initiative paradigm. The environment supports designers according to various possible requests of use: there are cases when the designer wants to have as much automatic support as possible, in other cases they may want to change some general design assumptions, yet in others they want to have full control in order to modify all the possible details in the design process.

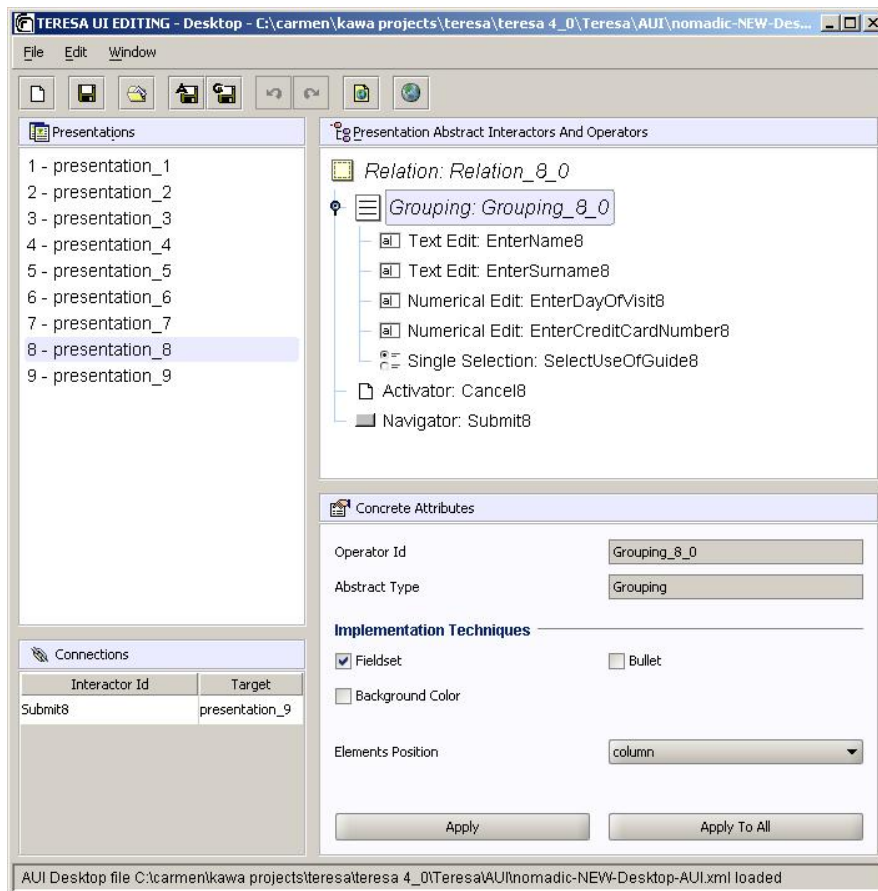


Figure 3 Panel for setting properties of composition operators

An example of the levels of control available in TERESA is shown in Figure 3 where you can see a window through which the designer can customise the presentations that are going to be generated by the tool. More in detail, in the top-left part of the window (the “*Presentations*” panel) the designer has a global picture of the current state of the design in terms of abstract presentations currently generated. The information visualised in the other panels of the window changes according to the currently selected presentation: in the bottom-left panel (“*Connections*”) the set of interactors allowing for moving from the currently selected to different presentations is visualised, whereas in the top-right part (“*Presentation Abstract Interactors and Operators*”) the different interactors contained in the currently selected presentation are shown, together with the related composition operators. In the bottom-right part of the window (“*Concrete Attributes*”) the possibility of selecting the specific communication technique to be used for implementing the

selected component of the abstract user interface (for the currently selected platform) is shown.

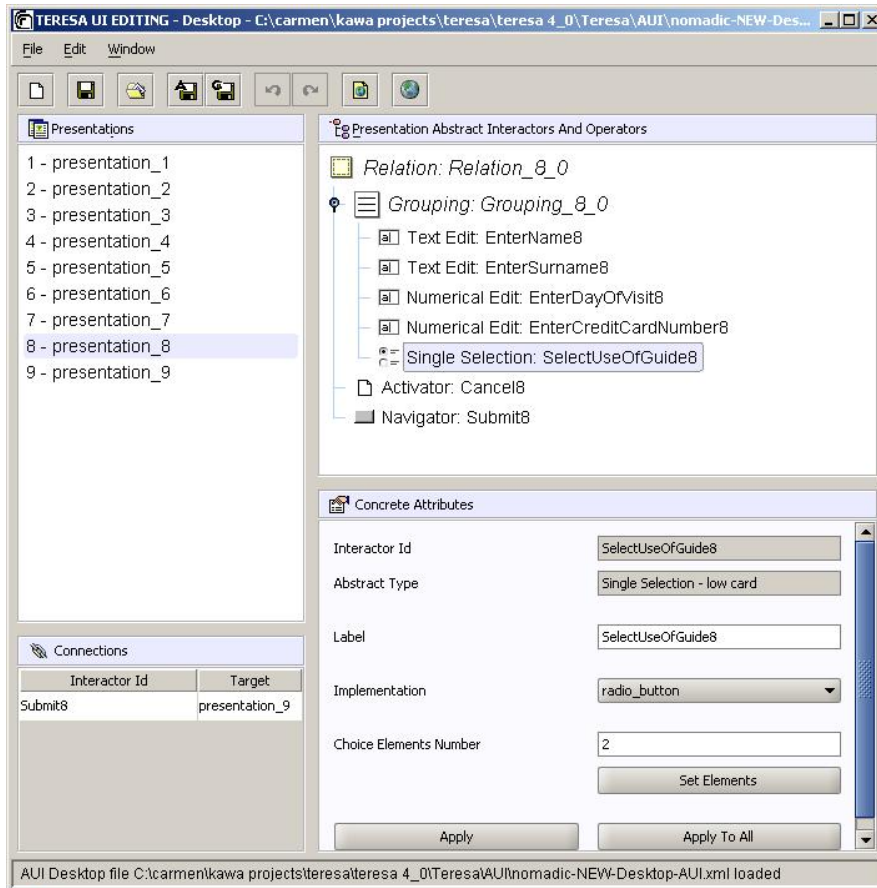


Figure 4 Panel for setting properties of the various interaction objects

The tool can provide suggestions according to predefined design criteria, but developers can modify them. For example, they can decide to implement the grouping operator by means of a fieldset (see Figure 3), the hierarchy operator through different font sizes, the ordering by means of an ordered list, and the relation operator by means of a form. However, these choices can be changed at any time. For example, if we consider the hierarchy operator: in the desktop environment it can be effectively implemented by varying the space allotted to the different objects in the presentation (for graphical user interfaces) or varying the size of text if a textual interaction object is considered. Neither of them can be used in the mobile environment respectively because in the first case the small area of cellphones' displays does not allow considering this dimension and, in the second case, the

limited capability of this platform does not allow the designer to vary too much the dimension of the text without compromising the quality of the result. In the vocal platform different levels of importance can be expressed increasing or decreasing the volume.

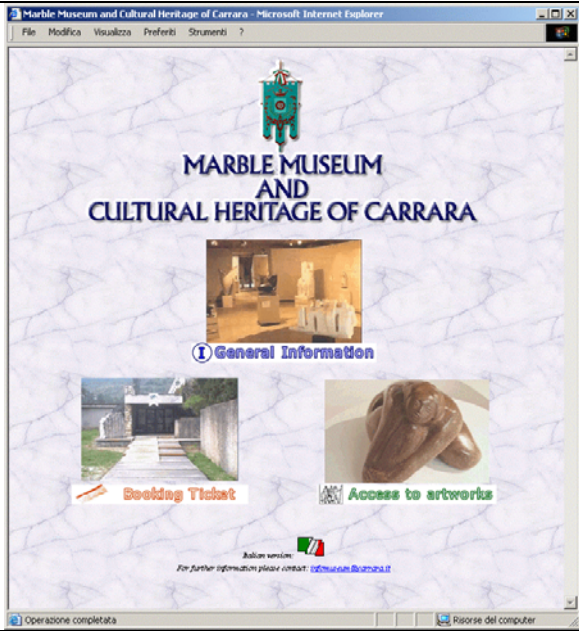
Desktop System	
	
Cellphone	VoiceXML-Enabled Phone
	<p>System:</p> <p>“Welcome to the Marble Museum Voice Response System. This service recognises your speech to provide you with the information you request. <i>(grouping sound)</i> If you would like some general information, say ‘information’, if you would like information about specific artworks, say ‘artworks’; if you would like to book a ticket, say ‘ticket’ <i>(grouping sound)</i>”</p>

Figure 5 Example fo same application with different interfaces depending on the interaction platform.

In addition, other differences can be found in supporting the user interface design for different platforms. For example, in the global parameters that are available to designers for customising the user interface: in the desktop system

parameters such as the background picture, the colour of the text, etc. are available, whereas in vocal devices they can be used to define welcome messages, use of barge-in options, synthesis and recognition properties. In the prototyping phase, the designer can select any presentation and change how to implement a specific interaction object through modifying some of its attributes (see Figure 4). It is worth pointing out that the tool enables saving the current configuration settings for future uses and modifications, so that the designers can incrementally build the user interface.

Moreover, the tool also supports variability within an interaction platform. For example, there are many types of devices that belong to the mobile phone platform. They can vary in terms of screen size, number and location of softkeys, colour support and so on. Thus, the tool supports the possibility of indicating the main characteristics of the device considered within the selected platform (such as number of characters per line or number of lines supported by the display). This further information is considered in the final generation of the user interface, for example, to decide whether to use field sets or images.

In Figure 5 we show some examples of user interfaces derived by applying the described method to a museum application. More specifically, the presentations refer to a situation in which the user wants to access information contained in the Marble Museum of Carrara town. As you can note, there are some differences concerning the presentations on the different platforms, the most evident one is represented by the different implementations of the abstract user interface grouping operator on the various platforms: on the desktop platform a list of graphical buttons is used, whereas on the cellphone a list of bullets is visualised, whereas in the vocal interface there are sounds that delimit the grouped elements.

6. Conclusions and Future Work

The TERESA environment supports design and development of multi-platform user interfaces through a number of transformations that can be performed either automatically or through interactions with the designer. To this end, a number of XML languages that capture the relevant information at different abstraction levels have been introduced. This allows designers and developers of ubiquitous services to concentrate on more semantic aspect without having to learn a lot of implementation languages and details.

The tool has provided a good opportunity to clarify various issues associated with the linkage between different models and the associated transformations, which must be fully understood in order to achieve real solutions and for which previous work in the area provided rather vague solutions. The tool can be freely downloaded at <http://giove.isti.cnr.it/teresa.html>.

While the current TERESA version supports the design and development of graphical and vocal interfaces for various platforms (currently through the generation of XHTML, XHTML Mobile Profile and VoiceXML, though other languages are planned), further work will be dedicated to supporting a broader set of modalities and their combinations.

Acknowledgements

We thank Francesco Correani for help in the tool implementation. We gratefully acknowledge support from the EU IST CAMELEON project (<http://giove.isti.cnr.it/cameleon.html>) and the EU SIMILAR NoE (www.similar.cc).

References

1. Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J. UIML: An Appliance-Independent XML User Interface Language, Proceedings of the 8th WWW conference, 1999.
2. R. Bandelloni, S. Berti, F. Paternò "Mixed-Initiative, Trans-Modal Interface Migration", Proceedings Mobile HCI 2004, Glasgow, September 2004, Lecture Notes Computer Science 3160, pp.216-227, Springer Verlag.
3. F. Correani, G. Mori, F. Paternò, Supporting Flexible Development of Multi-Device Interfaces, Proceedings EHCI-DSVIS'04, Hamburg, July 2004, Springer Verlag, Lecture Notes Computer Science.
4. G. Mori, F. Paternò, C. Santoro, "CTTE: Support for Developing and Analysing Task Models for Interactive System Design", IEEE Transactions on Software Engineering, pp.797-813, August 2002, Vol. 28, No. 8, IEEE Press.
5. Mori, G. Paternò, F. Santoro, C. Design and Development of Multi-Device User Interfaces through Multiple Logical Descriptions, IEEE Transactions on Software Engineering, Vol.30, N.8, August 2004, pp.507-520, IEEE Press.
6. Mullet, K., Sano, D. Designing Visual Interfaces. Prentice Hall, 1995.
7. Paternò, F. Model-Based Design and Evaluation of Interactive Application. Springer Verlag, ISBN 1-85233-155-0, 1999.
8. Paternò, F., Leonardi, A. A Semantics-based Approach to the Design and Implementation of Interaction Objects, Computer Graphics Forum, Blackwell Publisher, Vol.13, N.3, pp.195-204, 1994.
9. Puerta, A., Eisenstein, J. XIML: A Common Representation for Interaction Data, Proceedings ACM IUI 2001, pp.214-215.