

Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions

Giulio Mori, Fabio Paternò, and Carmen Santoro

Abstract—The increasing availability of new types of interaction platforms raises a number of issues for designers and developers. There is a need for new methods and tools to support development of nomadic applications, which can be accessed through a variety of devices. This paper presents a solution, based on the use of three levels of abstractions, that allows designers to focus on the relevant logical aspects and avoid dealing with a plethora of low-level details. We have defined a number of transformations able to obtain user interfaces from such abstractions, taking into account the available platforms and their interaction modalities while preserving usability. The transformations are supported by an authoring tool, TERESA, which provides designers and developers with various levels of automatic support and several possibilities for tailoring such transformations to their needs.

Index Terms—Design tools and techniques, user interfaces, heterogeneous clients, multiplatform user interfaces, authoring environments, abstract user interfaces, user interface design, task models.

1 INTRODUCTION

RECENT years have seen the ever-increasing introduction of new types of interactive devices (devices that support interaction with users). A wide variety of new interactive platforms is offered on the mass market. 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.

The availability of such platforms has forced designers to strive to make applications run on a wide spectrum of computing devices in order to enable users to seamlessly access information and services regardless of the device they are using and even when the system or the environment changes dynamically. On the one hand, this resulted in a dramatic improvement for the activities of users; on the other hand, it has radically changed the nature of many interactive applications, converting them to nomadic applications, namely, applications supporting user access in various contexts through different interactive devices. In fact, in order to guarantee a high level of user satisfaction, it is necessary that the applications should be able to adapt their user interfaces to the different context of uses, in particular, to the different devices used to access their functionality. This raises the fundamental issue of how to assist software designers and developers in building such applications, with the consequent need for novel methods and tools for the development of interactive software systems able to adapt to different targets while preserving

usability. Calvary et al. have used the term plasticity to indicate this type of user interface [7]: Our TERESA tool is a concrete solution to achieve such requirement.

In current practice, the design and development of multiplatform interfaces is often obtained through the development of several versions of the same application (one for each platform considered) that can at most exchange data. This solution, with no tool support to address multiplatform issues, is rather limited because it implies high implementation and maintenance costs. The opposite solution, completely automatic, is to use transcoding where an application written in a language for a platform is automatically transformed into an application in a language for another platform (see [11] for an example of HTML-to-WML transcoding). Strengths and weaknesses of different transcoding approaches (direct, hybrid, etc.) have been evaluated in [12], which provides useful hints to select the “best” technique depending on the current configuration. However, such an evaluation does not solve the traditional problem of such approaches which assume that the same tasks are supported by each platform and tend to support them in the same manner without taking into account the specific features of the platform at hand, so providing poor results in terms of usability.

Another solution proposed is the use of style sheets. Each platform is associated with a different set of style sheets. Thus, the same elements are presented differently according to the type of platform available. This could be useful, although it is still not capable of covering the wide range of possibilities that might occur when relationships between tasks and platforms (supporting tasks’ performance) are considered. Indeed, style sheets can help only try to better support the same tasks through different platforms, but unfortunately this is not always adequate because users often want to carry out different tasks according to the various types of platform. In addition, even mutual relations among tasks performed through several platforms may exist.

• The authors are with Institute of Science and Information Technology, National Research Council, Via G. Moruzzi 1, 56124 Pisa, Italy.
E-mail: {giulio.mori, fabio.paterno, carmen.santoro}@isti.cnr.it.

Manuscript received 29 Aug. 2003; revised 19 Feb. 2004; accepted 13 May 2004.

Recommended for acceptance by A. Mili.

For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0125-0803.

A more comprehensive solution can be obtained through the use of model-based approaches, which aim to support development through the use of meaningful abstractions to avoid dealing with low-level details. In the context of multidevice interface design and development, different abstraction levels can capture relevant information without having to address the plethora of details related to each device. This allows a tool to generate the specific version adapted for each device and modality, thus representing a viable alternative to overcome the limitations of other approaches.

In the next section of the paper, we describe the basic concepts characterising our approach and discuss related work. Then, in Section 3, we introduce our method. Section 4 is dedicated to describing how we have modified a previously developed tool for task modeling in order to support this new approach. The following section is devoted to describing the method and the tool (TERESA) that has been designed and implemented to support it (we discuss its requirements and the XML languages describing the various levels of abstraction considered). The transformations supported by the tool are described in more detail in Sections 6 and 7. After that, we report on experiences of use and lessons learned (Section 8); some concluding remarks along with indications for future work are provided in Section 9.

2 DESIGN OF MULTIPLATFORM APPLICATIONS

2.1 Basic Concepts

Our approach can be summarized in four words: *One Model, Many Interfaces* [22]. This means that we start with an abstract description of the activities to support and we are able to obtain different user interfaces for each available platform. In particular, we start with a task model of a nomadic application, which describes the activities that should be supported in order to reach the user's goals through different devices. Then, we allow designers to obtain effective user interfaces for the various platforms considered through a number of transformations implemented by our tool TERESA (Transformation Environment for inteRactive Systems representAtions).

In order to address the issues highlighted in the previous section, it is important to consider the various levels involved in an interactive system:

- *Task and object model.* At this level, the logical activities that need to be performed in order to reach the users' goals are considered along with the objects that have to be manipulated for their performance. Often tasks are represented hierarchically along with indications of the temporal relations among them and their associated attributes.
- *Abstract user interface.* In this case, the focus shifts to the interaction objects supporting task performance. An abstract user interface is defined in terms of a number of abstract presentations, each of them identifying the set of user interface elements perceivable at the same time. Each abstract presentation is composed of a number of *interactors* [21], which are abstract interaction objects identified in

terms of their semantics (the basic task they support).

- *Concrete user interface.* At this point, each interactor is replaced with a concrete interaction object that depends on the type of platform and media available and has a number of attributes that define more concretely its appearance and behavior.
- *Final User interface.* At this level, the concrete interface is translated into an interface defined by a specific software language (e.g., XHTML, Java, etc.).

To better understand such levels, we can consider an example of task: making a flight reservation. This task can be decomposed into selecting departure and arrival towns and, optionally, selecting seat and meal preferences. At the abstract user interface level, we need to identify the interaction objects necessary to support such tasks: for example, specifying departure and arrival towns calls for interactive selection objects. When we move on to the concrete user interface, we need to consider the specific interaction elements supported by the platform. So, in a desktop interface, the selection can be supported by a list object. This choice is more effective than a check-box because the list supports a single selection from a potentially long list of elements whereas the check-box is suitable to support multiple choices from a limited number of possibilities. The user interface of the example is the result of these choices and others involving attributes such as the font type/size, the foreground/background colors, and decoration images, and will be implemented in a specific language.

An early version of our approach and tool was already introduced in [14]. In this paper, we are able to present a novel engineered solution supported by a number of XML-based representations and transformations with various levels of automation. We also present the new associated tool.

2.2 Tasks and Multiplatform Environments

In order to identify the possible design solutions when multiple platforms are considered, we have developed a taxonomy of the relations between tasks and platforms. This is based on the observation that it is neither possible nor desirable to do everything through every platform. Each platform should be associated with specific contexts and should be effectively used only when they occur.

More precisely, in our taxonomy we identify various cases:

- *Same task on multiple platforms in the same manner.* For example, entering login and password is performed similarly in various platforms.
- *Tasks meaningful only on a single platform type.* For example, it is possible to use a desktop system to browse video trailers or access large attachments of email messages but these activities are not possible with small mobile phones. Conversely, the mobile phone can enable tasks that depend on the current position of mobile users (such as find an alternative route in case the current road is blocked), which are not meaningful for a desktop system.

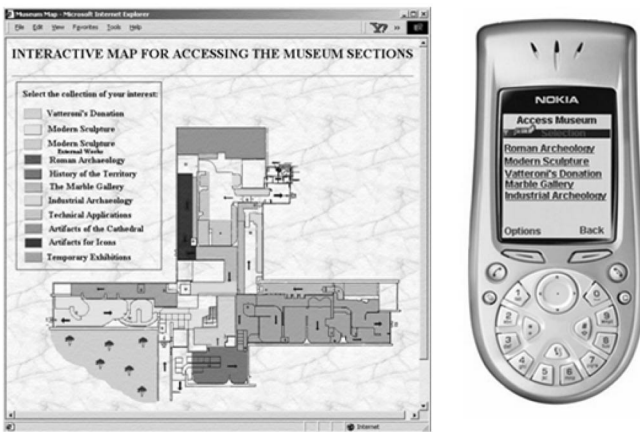


Fig. 1. Example of different interfaces supporting the same task through different platforms.

- *Dependencies among tasks performed on different platforms.* This occurs when the performance of a task through a platform enables or disables the performance of tasks through another platform. For example, during a city tour users can select a number of works of art. This, in turn, can enable the access to more detailed information regarding them through the desktop system.
- *Same task on multiple platforms but performed in different manner:*
 - *With different domain objects.* According to the resources available in a platform, different levels of detail can be provided regarding an argument. This means that some domain objects can be considered only if, for example, there is enough screen space available.
 - *With different user interface objects.* The same task can be supported through different interaction objects according to the media and the resources available. Fig. 1 shows how the *select museum section* task can be performed differently: using a graphical selection through a map in a desktop system (left part of the figure) and using a list of names in a mobile phone where a small screen is available (right part).
 - *With different task decomposition.* It occurs when the structure of a task needs to be changed. For example, if users want to reserve a flight seat, some systems will allow them to specify only the basic parameters (day, departure, and arrival towns), while others might allow for specifying a number of other (optional) parameters, such as preferred departure and arrival time, food preferences, and so on.
 - *With different temporal relations among subtasks.* In particular, the type of platform considered can impose specific constraints that do not occur in a more powerful system. So, for example, the small screen of a phone interface may require distributing among different sequential presentations some interaction techniques that in a desktop system can be included in a single

presentation and performed in any order. Another example is the vocal interface that serialises interactions that can occur concurrently in a graphical interface.

2.3 Related Work

The most common model-based approach in software engineering, UML [5], has paid very little attention to supporting the design of the interactive component of a software artefact. Specific model-based approaches have been developed to support user interface designers. The first generation of work in this area mainly focused on how to use models to only support development of desktop interactive applications, examples are Mobi-D [24] and Mastermind [27] or how to use such models to support user interaction at runtime, still in desktop applications [26]. As Myers et al. [16] pointed out, the increasing availability of new interaction platforms has raised a new interest in this approach in order to allow developers to define the input and output needs of their applications, vendors to describe the input and output capabilities of their devices, and users to specify their preferences. Then, a model-based system can choose appropriate interaction techniques taking into account all of these aspects. Even recent W3C standards, such as XForms [28], have introduced the use of abstractions to address new heterogeneous environments. In particular, XForms aims to separate presentation from content through the definition of a set of platform-independent, general-purpose, and focussing on the goal (or intent) behind each form control. In fact, the list of XForms control includes objects like *select* (choice of one or more items from a list), *trigger* (activating a defined process), *output* (display-only of form data), *secret* (entry of sensitive information), etc., rather than refer to concrete examples like radio buttons, checkboxes, and so forth, which are hard-wired to specific representations of such controls. This kind of logical description locates the types of abstractions supported by XForms at the abstract user interface level. However, the task level is not explicitly addressed. Once XForms is actually supported by the major browsers we plan to extend our environment in order to generate code in this mark-up language as well.

The new challenges have raised the need to define XML-based languages for representing the relevant concepts and ease their automatic manipulation. An example of a language that has addressed these issues is the *User Interface Markup Language (UIML)* (<http://www.uiml.org/>) [1], an XML-compliant language that supports a declarative description of a user interface in a device-independent manner. This has been developed mainly by Harmonia. However, their tools do not support the task level. Some research work on how to integrate task models with UIML has recently started at Virginia Tech [2], but its results are still preliminary and have not been incorporated in the Liquid environment supporting UIML. *The eXtensible Interface Markup Language (XIML)* (<http://www.xml.org/>) [25] is an extensible XML-based specification language for multiple facets of multiple models in a model-based approach. This has been developed by a forum headed by RedWhale software. A simple notion of task models is supported by this approach, for which tool support is not currently available.

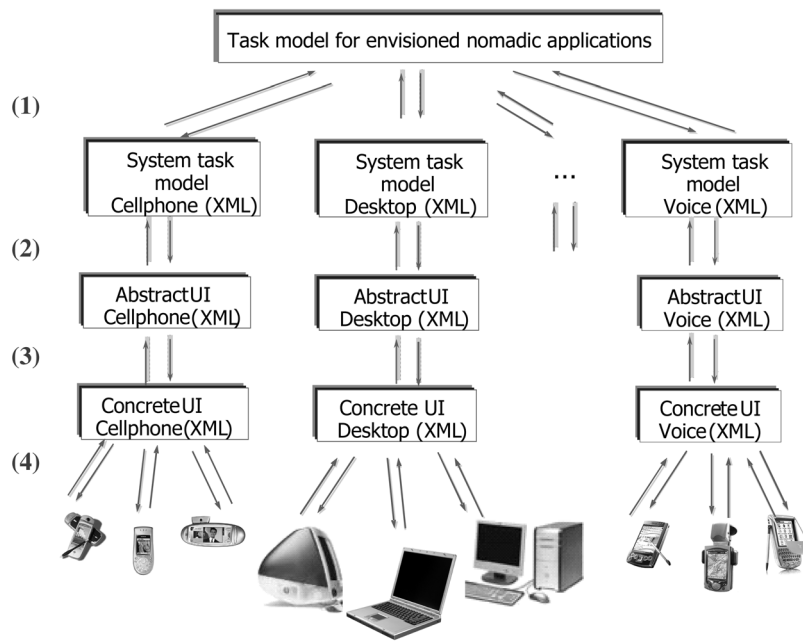


Fig. 2. The *One Model, Many Interfaces* approach.

While these approaches have shown some interesting results, there is still a lack of general solutions able to support the various relevant abstraction levels.

PUC (Personal Universal Controller) [17] is an environment that supports the downloading of logical descriptions of appliances and the automatic generation of the corresponding user interfaces. The logical description is performed through templates associated with design conventions, which are typical design solutions for domain-specific applications. The application area of this approach is oriented to the home domain and task models are not considered. Aura [9] is a project whose goal is to provide an infrastructure that configures itself automatically for the mobile user. When a user moves to a different platform, Aura attempts to reconfigure the computing infrastructure so that the user can continue working on tasks started elsewhere. In this approach, tasks are considered as a cohesive collection of applications. Suppliers provide the abstract services, which are implemented by just wrapping existing applications and services to conform to Aura APIs. For instance, Emacs, Word, and NotePad can each be wrapped to become a supplier of text editing services. So, the different context is supported through a different application for the same goal (for example, text editing can be supported through MS Word or Emacs depending on the resources of the device at hand). Other systems that have addressed cross-platform UI generation but that neither support task-level descriptions nor provide authoring environments are ICRAFT [23] and XWeb [18].

Another type of approach is the use of reverse engineering techniques to obtain an abstract description of an existing interactive system for a given platform and then use it as a starting point for a new design adapted for a new platform. Examples of such reverse engineering approaches are Vaquita [6] and WebRevEng [19]: They both start with a desktop Web site code, the former allows designers to

obtain an abstract user interface whereas the latter is able to derive the correspondent task model. Both of them can be used as complementary support for our approach: Once an abstract description has been obtained, our tool can help the developer to obtain a new design suitable for a different type of platform.

3 THE METHOD

In this section, we introduce our method for model-based design of multiplatform user interfaces. It is composed of a number of steps that allows designers to start with an envisioned overall task model of a nomadic application and then derive concrete and effective user interfaces for multiple devices. In Fig. 2, the different data managed at each level have been specified within rectangles, whereas the various steps performed among such levels have been referred through numbers. The next sections describe in detail each step.

We start with *High-level task modeling of a multiplatform application*. In this phase, designers develop a single model, which addresses the possible contexts of use and the various platforms involved, including a domain model aiming to identify all the objects that have to be manipulated to perform tasks and the relations among such objects. The purpose of this model is to provide an overview of the tasks supported by the nomadic application. For each task, it is possible to indicate what platforms are able to support it and it is also possible to show dependencies among tasks that can be performed through different platforms.

The next phase (see Step 1 in Fig. 2) is *developing the system task model for the different platforms considered*. Here, designers have to filter the nomadic task model according to the target platform and, if necessary, further refine the resulting task model depending on the specific platform considered, thus obtaining the various platform-dependent

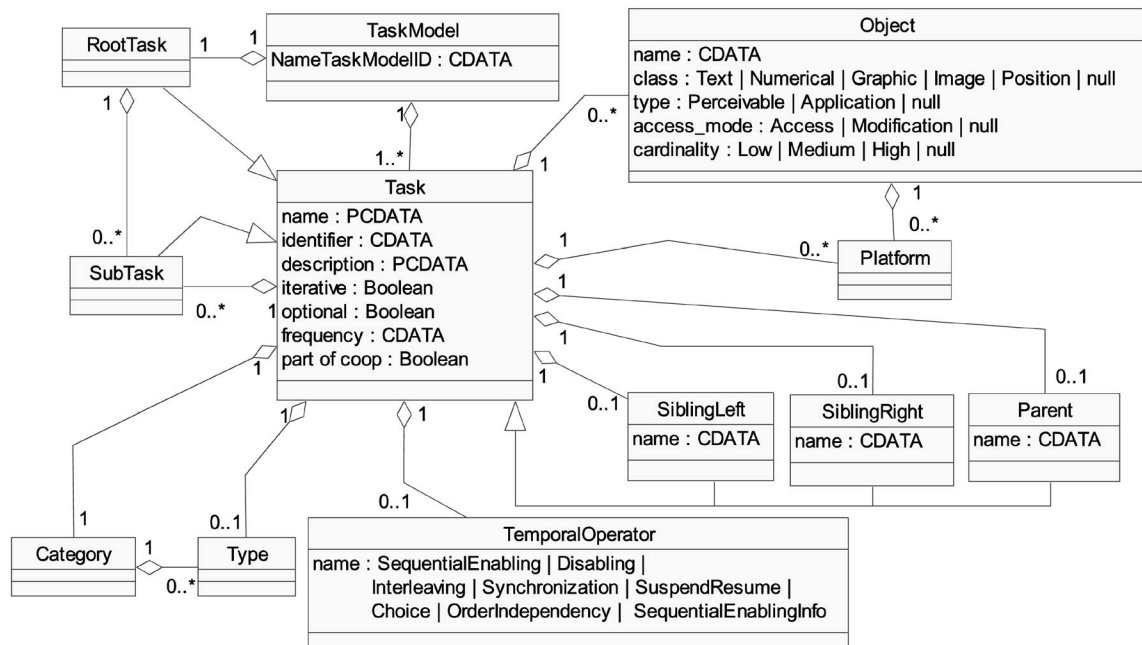


Fig. 3. Class diagram representing the concepts of the ConcurTaskTrees notation for task models.

task models, which represent the input of the next step. Then, designers have to move *from the system task model to the abstract user interface* (see Step 2 in Fig. 2). The goal of this phase is to obtain an abstract description of the user interface composed of a set of abstract presentations that are identified through an analysis of the task relations. Each presentation will be specified by means of abstract interaction objects composed through various operators (grouping, ordering, hierarchy, relation), which stand for different composition techniques (for example, the grouping operator will highlight the fact that there are objects that should be grouped together because they are closely related to each other). The next step (transformation (3) in Fig. 2) is *from abstract to concrete interfaces*. This phase is completely platform-dependent and has to consider the specific properties of the target platform. Then, every interactor is mapped into interaction techniques supported by the particular target platform, and the abstract composition operators also have to be appropriately associated with techniques highlighting their logical meaning: A typical example is the set of techniques for conveying grouping relations in desktop visual interfaces by using presentation patterns such as proximity, similarity, and color.

The last phase (Step 4 in Fig. 2) is *Code generation*, where the code is generated starting with the concrete interface description in the target software environment. This can be done completely automatically because all the design choices have already been made. In case there is a need for implementations in different languages for the same platform only this transformation needs to be changed.

At any time, it is possible to go back in this sequence of transformations in order to revise the previously considered models in case new issues have been identified after performing a transformation.

In order to provide tool support for this method, we have defined XML languages for the task model, the abstract

level, and the concrete level; we extended a tool for task modeling we already implemented; and we have developed from scratch the TERESA tool for authoring multiplatform applications. Both tools will be described in the next sections.

4 TASK MODELS OF NOMADIC APPLICATIONS

The ConcurTaskTrees Environment (CTTE) [13] is an engineered, publicly available (<http://goive.isti.cnr.it/ctte.html>) tool for task modeling. It eases the development of task models described using the ConcurTaskTrees (CTT) notation [20] and supports their analysis through a number of features (metrics evaluation, interactive simulation, etc.). The ConcurTaskTrees notation supports a hierarchical description of task models with the possibility of specifying a number of temporal relations among them (such as enabling, disabling, concurrency, order independence, and suspend-resume). In addition, for each task, it is possible to specify what objects need to be manipulated for its accomplishment (it is possible to consider both user interface and domain objects), as well as a number of additional attributes (such as frequency) (see Fig. 3).

In order to be able to capture the specific aspects of task models for nomadic applications, we needed to extend CTTE in the following way:

- The platform attribute has been added in each task specification; its purpose is to indicate the types of platforms that are suitable to support a task. It is worth noting that at this level—the task level—sets of devices sharing certain similarities are considered, rather than *specific* devices. So, in our framework, we provide for typical sample device clusters as mobile phones and PDAs are, together with the possibility

for designers to define their own platforms. This has proven to be both feasible and flexible to tackle the problem of dealing with the disparate devices that our approach has to consider. Nevertheless, additional levels of refinement within the same cluster are considered in the last phase of the method, when knowing the specific characteristics of the devices considered becomes useful for producing effective final user interfaces.

- The platform attribute has also been associated with the objects manipulated during task accomplishment. Indeed, CTT allows designers to specify for each task what objects should be manipulated during its performance.
- The filtering functionality according to the platform attribute has been added. This means that it is possible to take a task model of a nomadic application and ask to view only the parts that can be actually supported by a given platform. If the task model has not been carefully designed, this can generate a model with some unconnected parts. However, the tool can help in refining it. The filter-and-refine mechanism allows designers to maintain a global picture of the application and derive the corresponding model for each platform.

5 TERESA

5.1 Approach

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. This is important to obtain a tool able to satisfy a variety of needs: situations when the time available is short, the application domain is rather narrow, or the designer has no expertise call for completely automatic solutions. When designers are expert or the application domain is either broad or has specific aspects, then more interactive environments are useful because they allow the designer to directly make important design decisions.
- *Model-based.* As Myers et al. pointed out [16], 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.
- *Multiple logical levels described through XML-based languages.* XML-based languages ease the possibility of importing and exporting user interface descriptions through different tools and environments, they can be used to describe each relevant logical description.
- *Top-down.* This approach is an example of forward engineering. Various abstraction levels are considered, and we support cases when designers have to

start from scratch. So, they first have to create more logical descriptions and then move on to more concrete representations until they reach the final system. We are aware that in other cases bottom-up approaches may be preferable, for example, when designers aim to redesign a desktop Web site for a mobile device. To this end, we already had experiences of integrating our tool with a reverse engineering tool (Vaquita) that built the concrete description of an existing application that was redesigned for a mobile phone through TERESA.

- *Different entry-points.* Our approach aims to be comprehensive and to support the various possibilities indicated by our task/platform taxonomy, although it may happen that only a part of it needs to be actually supported (for example, when only different brands of mobile phone are considered). In this case, there is no need for a nomadic task model, given that only one type of platform is involved and designers can start with either the corresponding system task model or the corresponding abstract user interface.
- *Web-oriented.* The Web is everywhere and, so, we decided that Web applications should be our first target. However, the approach is also valid for generating user interfaces for other types of software environments, such as Java applications, Microsoft environments, etc. This simply requires extending the implementation of the last transformation (from the concrete to the final user interface) for the specific target software.

5.2 Main Functionality

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 XML 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 for 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 XML 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

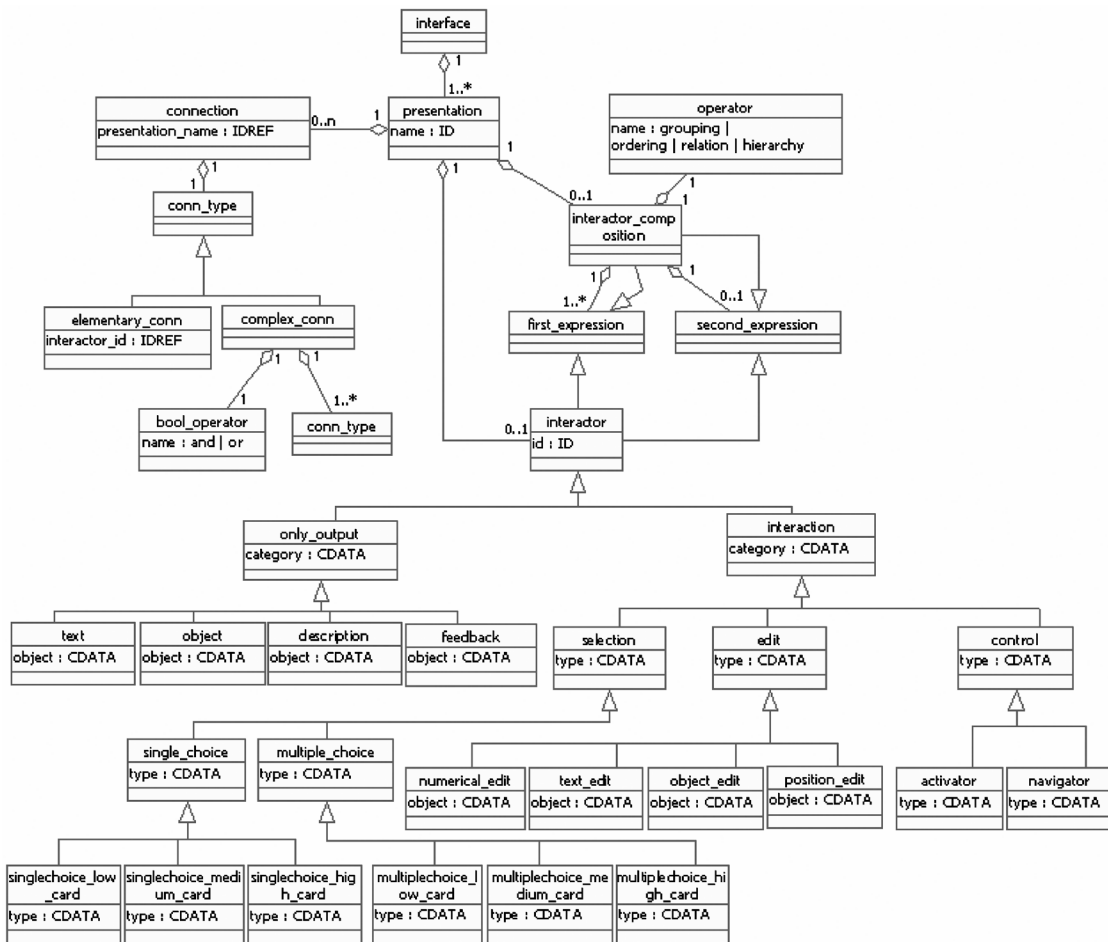


Fig. 4. The concepts and their relations represented in the TERESA notation for abstract user interfaces.

interface, in terms of both its static structure (the “presentation” part) and dynamic behavior (the “dialogue” part), is saved for further analyses and transformations. It is worth pointing out that by using TERESA it is also possible to 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.

- *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.

5.3 TERESA Abstract User Interface Language

An abstract user interface is composed of a number of presentations and connections among them. Each presentation defines a set of presentations and interaction techniques perceivable by the user at a given time. The connections define the dynamic behavior 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 complex conditions in case a specific combination of interactions should trigger the change of presentation.

The structure of the presentation is defined in terms of interactors (abstract descriptions of interaction objects classified depending on their semantics) [21] and their composition operators (see Fig. 4). 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), and control (to trigger an event within the user interface, which can be useful to activate either a

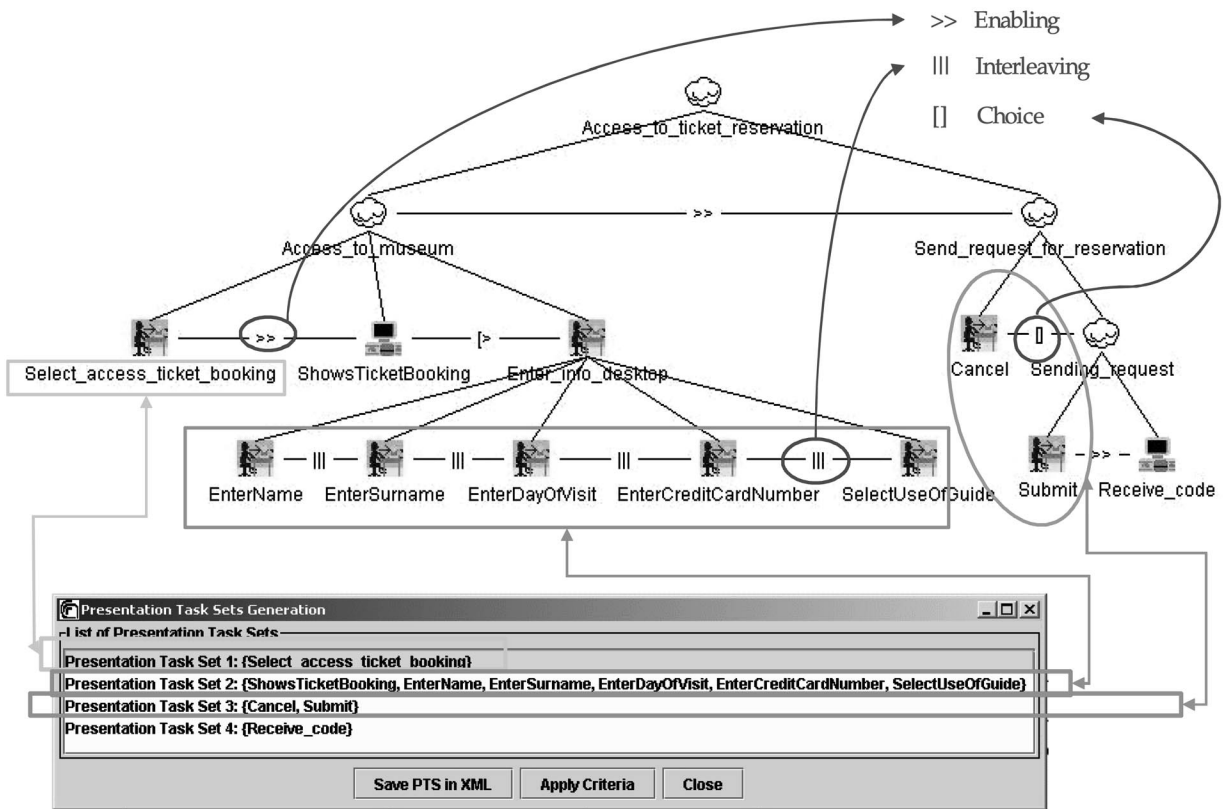


Fig. 5. Example of calculation of Presentation Task Sets.

functionality or the transition to a new presentation). Differently, an `only_output` element defines an interactor that 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 an interactor or a composition of several ones. 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 [15]. 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.

6 FROM SYSTEM TASK MODELS TO ABSTRACT USER INTERFACES

6.1 Identification of the Presentation Task Sets

The Presentation Task Sets (PTSs) are sets derived from a CTT task model and represent tasks that are enabled over

the same period of time. In particular, the PTSs are derived by analyzing the formal semantics of the CTT temporal operators: For example, if two tasks are executed concurrently, they are enabled at the same time, so they belong to the same set. Alternatively, if two tasks are connected through an enabling operator, the second task will be executed just after the first one, so the tasks do not belong to the same Presentation Task Set. Fig. 5 shows an example of a set of PTSs associated with an excerpt of task model concerning a museum application. Depth-first visiting the task tree, the first leaf subtask (*Select access ticket booking*) is followed by an enabling operator (`>>`). This means that no other task is enabled together with it, so there is one presentation task set composed of only such task. Then, there is an application task (*ShowsTicketBooking*) which is connected with a disabling operator (`[>`) with a group of tasks concurrently executed (see the interleaving operator “`|||`” between *Enter name*, *Enter surname*, *Enter Day Of Visit*, *Enter CreditCard Number*, *SelectUseOfGuide*): Such concurrency implies that they are enabled over the same period of time and, only after performing all of them, the next tasks are enabled. Therefore, a distinct presentation task set is associated with them. After the users have entered their personal data, they will be able either to *Cancel* or *Submit* the request (note the choice operator `[]`). Since the user should be able to choose between one of them, such tasks are enabled over the same period of time and belong to the same set.

The task model shown in Fig. 5 describes the activities involved in reserving a ticket under the assumption that a

specific platform—the desktop in this case—is used. If another platform is considered, the specification of the same activity might change quite radically, e.g., due to a different organisation of the expected interactive session (for instance, entering surname and credit card number, etc., might be executed sequentially). Hence, another task model is specified accordingly, which in turn leads to a modified arrangement of PTSs.

If a nomadic task model is considered, then the same type of representation is provided. The main difference is that in this case it includes tasks that can be performed through different platforms and also indicates the relations among such tasks.

Automatic calculation of the presentation task sets implies defining the conditions that allow passing from one PTS to another, which depends on the temporal operators among the various tasks. Such conditions can be presented in various manners. One example is when one task alone allows for enabling the next set of tasks (the simplest example is when one task is connected through an enabling operator with other tasks). So, in this case, the condition coincides with the task itself. However, there are other cases in which such a condition reveals to be a Boolean expression involving two or more tasks. It happens when there is a group of tasks which are performed concurrently and this group is connected with an enabling operator to another set. In this case, whatever the execution order of those tasks, the condition for moving on to the next set of tasks is the completion of all the concurrent tasks. As a result, the Boolean condition expressing such a constraint is an AND operator applied to all the tasks involved. In other cases, the OR Boolean operator is necessary to express the fact that the performance of just one task in the set enables the next presentation. Moreover, also complex expressions with combinations of AND/OR operators can occur.

6.1.1 Heuristics for Obtaining Presentation Task Sets and Transitions

As the number of presentation task sets generated automatically from the task model is of the same order as the number of the CTT enabling operators appearing in the task model, a direct mapping between them and the user interface presentations might produce excessively modal user interfaces or a high number of presentations with a very limited number of elements. A number of heuristics have been identified for the purpose of helping designers to limit the number of presentations by merging two or more PTSs. The reasons for this step are to reduce the initial number of PTSs, which as previously noted can be very high in some cases and include significant information (such as data exchange is) within the same presentation, even when the tasks involved belong to different PTSs, so that users can better follow the flow of information. These rules are particularly useful when desktop systems are considered. Up to now, the heuristics that have been identified are the following:

- 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.

- If a PTS is composed of just one element, it can be included within another superset containing its element.
- 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 the same PTS.
- 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.

It is worth noting that the designer can decide about the heuristics' application, also taking into account the features of the specific platform considered. For example, if we consider graphical user interfaces, it is likely that, on devices with small screens, the heuristics will be less applied than on other devices with more extended capabilities. The reason is that desktop systems rely on large screen areas, whereas on small displays too many user interface objects in the same presentation would tend to add clutter rather than increase usability.

6.2 Mapping Tasks to Interactors

Once we have obtained the information about tasks belonging to each presentation task set together with the transitions among the various sets, the next step is obtaining the description of the abstract user interface, structured into presentations and connections among such presentations. Each presentation is identified by one Presentation Task Set and has the goal to support the associated tasks. Each presentation is defined in terms of interactors and operators. In order to do this, two steps are necessary:

- mapping the various tasks into corresponding interaction objects of the abstract user interface;
- deriving the appropriate composition operators that should be applied to the various interactors.

In the following two sections, further details will be provided about such steps.

6.2.1 Transforming Tasks into Abstract Interaction Objects

In order to map the various tasks into the related interactors, we have to consider the information contained in the task model. First of all, the allocation of a task (whether the task is performed either through an interaction between the system and the user, or just by the application) is useful information to identify the category of the associated interactor (respectively, Interaction or OnlyOutput). Each of these categories represents a set of interactors identified by the type of the task supported. For example, an “edit” task type indicates that the corresponding interactor should allow information modification, as well as a “selection” task type indicates that the associated interaction techniques should support the performance of this kind of activity.

Actually, these interactors are generic classes of interactors, which means that it is still possible to identify more

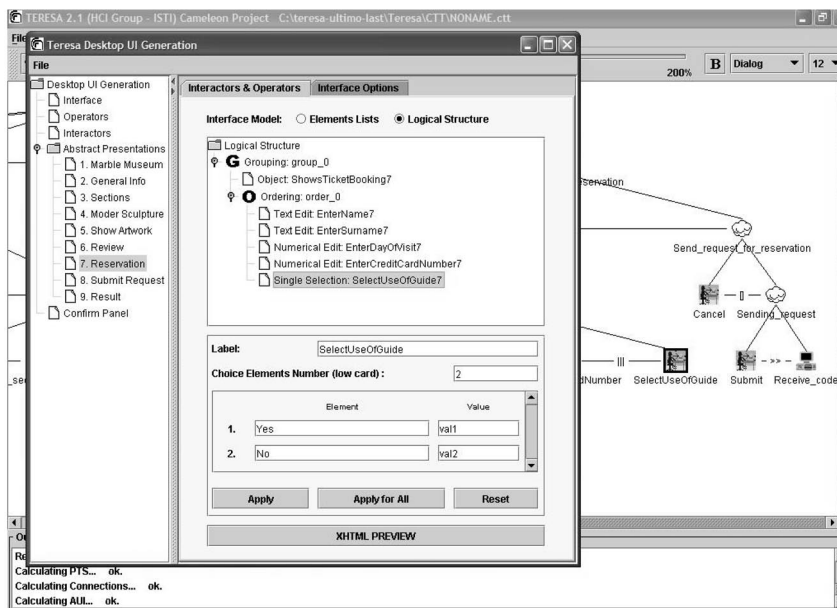


Fig. 6. The interface for managing the Abstract User Interface generated from the task model.

specialized subclasses up to reaching elementary interactors. The last step for deciding the type of interactor analyses the semantic effects of the interactions to support. In some cases, the combined analysis of task type and class of task objects manipulated will allow the identification of the specific, elementary interactor that is finally selected for the mapping. In other cases, different conditions might need to be specified because particular task types require the specification of further attributes. For instance, as far as selection tasks are concerned, besides specifying the type of interactor supported (single or multiple), in order to identify the appropriate type of interactor, we must also define the cardinality (high/medium/low) of the object set from which the selection will be made. The same type of rule is applied in the case of application tasks. “Feedback” and “Visualise” are examples of task types belonging to the application task category: They indicate the activity of presenting some results of a server-side application processing or some application data, so they will be mapped onto *only_output* interactors. Also, in this case, there are different interactors suitable to support these activities.

Fig. 6 shows the interface for handling the abstract user interface automatically generated by the tool from the corresponding task model. On the left side, there is the list of presentations that have been generated automatically. One of them has been selected (*Reservation*) and the tool displays the elements composing it and its logical structure on the right. This presentation corresponds to the PTS 2 in Fig. 5. There is an ordering of five elements and then a grouping of one element with the first ordering. For each composition operator, there is the list of the associated interactors with the indication of their types. When one interactor or composition operator is selected in the upper part of the window, then the associated concrete elements are shown in the bottom part. In the figure, a single selection element is selected and the lower part shows the associated label and how it is currently implemented. The

designers can modify it if they are not satisfied. The tool maintains links among elements at different abstraction levels, so that, if at some point there is a need to modify the upper level, it is easy to identify the part of the upper level corresponding to the elements under consideration. Fig. 6 shows how it is possible to select one interactor and have the tool automatically highlight the corresponding task in the task model with a bold border around its icon (the *SelectUseofGuide* task in the example).

6.2.2 Identification of the Interactor Composition Operators

The interactor composition operators that appear in the abstract user interface are derived by analyzing the CTT task model specification. In particular, not only are the CTT operators analysed, but also other attributes of the tasks (such as frequency) take part in this transformation.

In fact, on the one hand, when some conditions occur, several CTT operators are directly linked with operators of the abstract user interface. This is the case when two or more tasks sequentially performed end up in the same presentation task set: The sequencing at the task level can be translated, at the abstract user interface level, by applying the ordering operator.

On the other hand, other operators of the abstract user interface take into account the attributes of the involved tasks, rather than the CTT temporal operators existing among them. 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 an 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.

6.2.3 The Definition of the Links with the Functional Core

One issue is how to connect the interactive part of a software application with the functional core, the set of application functionalities independent of the media and the interaction techniques used to interact with the user. Since in the task model the activities supposed to be performed by the system are identified by the application tasks, the tool automatically identifies in the task model the two situations that are relevant for such a link:

- when (the system tasks associated with) internal functionalities are supposed to perform information access to the back-end;
- when an internal functionality needs to present information to the user.

In order to manage these two cases, in each task, specification the objects handled to perform the tasks are indicated, classified in *perceivable* and *application* objects: The first objects have a direct impact on the user interface inasmuch as they are associated with concrete interface elements appearing on the user interface (menus, buttons, images, labels, etc.), whereas the “application” objects refer to logical objects connected with the application underneath.

In the first situation, no perceivable activity on the user interface side occurs, so it is possible to automatically identify the related tasks because they are system tasks characterised by a lack of perceivable objects in their specification. The interactors involved in such functionality are those handling the application objects whose values should be used to send a request to the functional core. In the abstract interactor, there is information indicating what functionality of the core should be accessed (identified through the corresponding application task) and other attributes indicating the parameters associated to such a request. This information is further refined when moving to the concrete interface level.

Also, the second case (when an internal functionality needs to present information to the user) can be automatically detected because, in this situation, we have application tasks manipulating both perceivable and application objects. This means that the application functionality that has to present information to the user must communicate with the interactor handling the perceivable object associated with the task.

7 FROM THE ABSTRACT USER INTERFACE TO ITS IMPLEMENTATION

As mentioned in Section 5.1, one of the 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 and, 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. At the beginning, a number of general

parameters and information are presented: As you can see in Fig. 6, in the left part, the designer has a global picture of the current state of the design in terms of abstract presentations currently generated, general user interface parameters, composition operators settings, etc., together with the possibility of further selecting one of these options and visualizing/modifying the related attributes in the right-hand panel window. An example of the levels of control available in TERESA for designers is the possibility of selecting the specific communication technique to be used for implementing each interactor composition operator. The tool can provide suggestions according to pre-defined design criteria, but developers can modify them: For example, they can decide to implement the grouping operator by means of a fieldset, the hierarchy operator through different font sizes, the ordering by means of an ordered list, and the relation operator by means of a form.

In addition, depending on the type of platform considered there are different ways to implement design choices at the user interface level. For example, the same grouping operator can be implemented with different techniques depending on whether the desktop or the mobile or the vocal platform is considered. In fact, the desktop environment allows using tables, so the grouping operator can be implemented by a number of techniques including both unordered lists by row and unordered list by column (apart from classical grouping techniques like fieldsets, bullets, and colors). The small capability of a mobile phone does not allow implementing the grouping operator by using an unordered list by column then this technique is not available on this platform. In a vocal device, grouping can be achieved through inserting specific sounds or pauses or using a specific volume or keywords [4].

Other differences regarding the environments related to each platform can be found for the hierarchy operator: In the desktop environment, the hierarchy operator 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 the dimension of the text too much without compromising the quality of the result. In the vocal platform, different levels of importance can be expressed by increasing or decreasing the volume.

In addition, other differences can be found to support the user interface design between platforms, for example, in the global parameters that are available to designers for customizing the user interface: In the desktop system, parameters such as the background picture, the color 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 either how to implement a composition operator in it or a specific interaction object

TABLE 1
Presentations of Artwork Information on Different Platforms

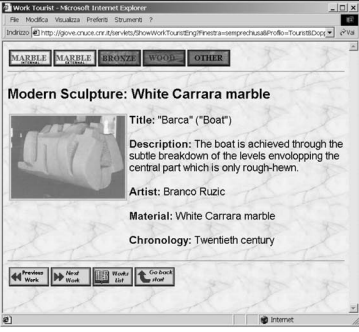
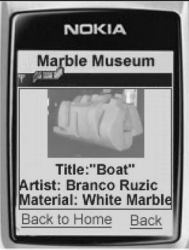

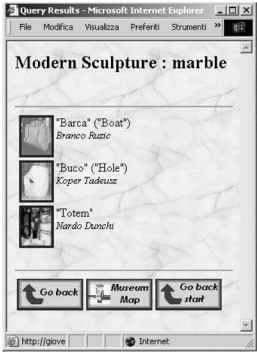



Desktop System	Cellphone	VoiceXML-Enabled Phone
 <p>The screenshot shows a web browser window with a title bar 'Work Tourist - Microsoft Internet Explorer'. The address bar shows 'http://gouve.govve.org/StarView/ShowWorkTourist.asp?WorkTouristID=5&WorkTouristDoc=TouristDoc...'. The main content area displays 'Modern Sculpture: White Carrara marble' with a title 'Barca' ('Boat'), a description, artist 'Branco Ruzic', material 'White Carrara marble', and chronology 'Twentieth century'. There are navigation buttons at the bottom: 'Go back', 'Go forward', 'Home', 'Print', 'Stop', 'Refresh', 'Back', 'Forward', 'Home', 'Print', 'Stop', 'Refresh'.</p>	 <p>The screenshot shows a Nokia mobile phone screen with 'Marble Museum' at the top. Below it is a small image of the artwork. Text below the image reads: 'Title: "Boat"', 'Artist: Branco Ruzic', 'Material: White Marble'. At the bottom are two buttons: 'Back to Home' and 'Back'.</p>	<p> System:</p> <p>“The ‘Boat’ has been achieved through the subtle divisions of the planes enveloping its central part, which is only rough-hewn; the material is white marble.” (Five second pause) “Remember that if you would like to return to the main menu, say ‘home’ or if you would like to go back to the previous menu, say ‘back’.”</p>

TABLE 2
Implementation of the Grouping Operator on Different Platforms

Desktop System	Cellphone	VoiceXML-Enabled System
 <p>The screenshot shows a web browser window with a title bar 'Query Results - Microsoft Internet Explorer'. The main content area displays 'Modern Sculpture : marble' followed by a list of artworks: 'Barca' ('Boat') by Branco Ruzic, 'Bucco' ('Hole') by Koper Tadeusz, and 'Totem' by Nardo Dunchi. At the bottom are navigation buttons: 'Go back', 'Museum Map', 'Go back start'.</p>	 <p>The screenshot shows a Nokia mobile phone screen with 'Marble Museum' at the top. Below it is the text 'Artworks:' followed by a list: 'Boat', 'Totem', 'Hole'. At the bottom is a button: 'Back to Home'.</p>	<p> System: (<i>grouping sound</i>) The artworks contained in the section are the following: ‘Boat’, ‘Totem’, ‘Hole’. If you would like information on one of these please say its name (<i>grouping sound</i>)</p> <p> Remember that if you would like to return to the main menu, say home.</p>

or some of its attributes. 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.

As we mentioned before, 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, color support, etc. 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 Table 1 and Table 2, we show some examples of user interfaces derived by applying the described method to a museum application.

More specifically, in Table 1 the presentations refer to a situation in which the user has selected information about a specific artwork of Modern Sculpture section. As observed, there are some differences concerning the presentation of

the artwork selected: On the desktop system, the picture of the artwork is shown, together with additional information (title, description, artist, material, etc.). On the cellphone, the picture is provided as well but the textual information is more concise; in a VoiceXML-enabled system, a vocal description of the artwork is provided. It is worth pointing out that, on the different platforms, the navigation mechanisms also change: On both the desktop system and cellphone, some links have been (visually) presented, although they differ in number. In fact, on the desktop system, there is enough room for displaying also links to other sections of the museum, whereas, on the cellphone, the choice is more limited. In the third case (VoiceXML), the navigation is implemented by dialogs that have been vocally provided.

Another interesting point is represented by the different implementations of the abstract user interface operators on the various platforms. In Table 2, the different implementations of the grouping composition operator on the various platforms have been shown: In the first two presentations, a set of graphical buttons (first case) or textual links (second case), all arranged vertically, is used, whereas, in the vocal

interface (third case), there are sounds that delimit the grouped elements.

8 EXPERIENCES OF USE AND LESSONS LEARNED

The tool has been used in a number of cases for various purposes in different settings. In a HCI course at a computer science department after a two-hour lesson on task modeling, students were introduced to the tool for developing and analyzing task models for another couple of hours. Next, they attended a lesson on model-based design and, after that, they were able to use the TERESA tool for the development of a small application running on a platform chosen by them. Each student developed the user interface for only one platform because the time available was limited but different students selected different platforms. So, at the end, it was possible to obtain a multiplatform application. This suggests that the environment requires limited effort to learn it and even people with low experience were able to generate user interfaces according to usability criteria thanks to the tool support. A more formal evaluation was conducted at the Motorola Italy software development centre [8]. The first experiment consisted in starting with a given task model and obtaining the corresponding user interface for both desktop and mobile phone. The exercise goal was to realize a simple version of an e-desk application accessible through desktop and mobile systems. A second experiment was conducted in order to collect more information about satisfaction and cost/effectiveness of the approach. The experiment consisted in developing a prototype version of an e-Agenda application running on both desktop and mobile phone and including the following functionalities: visualization of the appointments of a single day, visualization of the details of each appointment, possibility of inserting/modifying/deleting an appointment. The evaluators were required to collect quantitative metrics related to development efficiency, such as the total effort needed to complete the exercise expressed in terms of time required for the first version and the final version, and categorized by process phase, as well as the number of errors introduced. Results showed similar total times for the traditional and TERESA approaches, with different distributions over the development phases and between time required by the first and the final version. The TERESA-supported method offered a good support to fast prototyping, producing a first version of the interface in a significantly shorter time. On the other side, the time required to modify it resulted in an increase. The use of the tool almost doubled required time at redesign stage, while, at the development stage, the results showed dramatically improved prototyping performance, reducing needed time to half. This leaves a margin for further improvement since the design time required by TERESA approach is expected to decrease as the subjects become more familiar with model-based techniques and notations.

Moreover, the reported slight total time increase of using TERESA with respect to using traditional approaches (on average, it was half an hour) is acceptable since it involves a trade off with design overall quality: Many subjects appreciated the benefits of a formal process supporting the individuation of the most suitable interaction techniques. For example, designers reported satisfaction about

how the tool supported the realization of a coherent page layout and identification of links between pages. The evaluators noticed and appreciated the improved structure of the presentations and a more consistent look of the pages resulting from the model-based approach. This is also coupled with an increased consistence between the desktop and the mobile version, pointed out by almost all the evaluators.

In summary, TERESA emerged from the evaluation as an appealing solution for designing and developing user interfaces on multiple and heterogeneous devices.

9 CONCLUSIONS

We have presented a method and the associated tool supporting design and development of nomadic applications. While we think that designers should be aware of the potential platforms (not devices) early on in the design process, so they can identify the tasks suitable for each, the method allows developers 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.

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 modeling tool has now been integrated into TERESA so that designers can use just one tool and not lose time switching between two different tools. TERESA is publicly available at <http://goive.isti.cnr.it/teresa.html>.

The logical descriptions and the transformations defined in the method presented can also be used at runtime to support migratory interfaces (interfaces able to dynamically move from one device to another) [3]. 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. 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.

ACKNOWLEDGMENTS

The authors gratefully acknowledge support from the European Commission through the CAMELEON IST project (<http://goive.isti.cnr.it/cameleon.html>). They also thank Silvia Berti and Francesco Correani for the help in the implementation of the abstract-to-final interface transformation and the Motorola Italy Software Development Centre for useful feedback in the TERESA evaluation.

REFERENCES

- [1] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams, and J. Shuster, "UIML: An Appliance-Independent XML User Interface Language," *Proc. Eighth WWW Conf.*, 1999, http://www.harmonia.com/resources/papers/www8_0599/index.htm.
- [2] F. Ali, M. Perez-Qinones, and M. Abrams, "Building MultiPlatform User Interfaces with UIML," *Multiple User Interfaces*, A. Seffah and H. Javahery, eds., pp. 95-118, 2003.
- [3] R. Bandelloni and F. Paternò, "Flexible Interface Migration," *Proc. Intelligent User Interfaces (IUI '04)*, pp. 148-157, 2004.
- [4] S. Berti and F. Paternò, "Model-Based Design of Speech Interfaces," *Proc. Design, Specification, and Verification of Interactive Systems Workshop*, pp. 231-244, 2003.
- [5] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language Reference Manual*. Addison Wesley, 1999.
- [6] L. Bouillon and J. Vanderdonck, "Retargeting Web Pages to other Computing Platforms," *Proc. IEEE Ninth Working Conf. Reverse Eng. (WCRE '02)*, pp. 339-348, 2002.
- [7] G. Calvary, J. Coutaz, and D. Thevenin, "A Unifying Reference Framework for the Development of Plastic User Interfaces," *Proc. Eng. Human-Computer Interaction Conf.*, pp. 173-192, 2001.
- [8] C. Chesta, M. Fliri, S. Martini, B. Russillo, and C. Barbero, "First Evaluation of Tools and Methods," CAMELEON Project Document: D3.4, July 2003.
- [9] J. De Sousa and D. Garlan, "Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments," *Proc. IEEE-IFIP Conf. Software Architecture*, 2002.
- [10] J. Eisenstein, J. Vanderdonck, and A. Puerta, "Applying Model-Based Techniques to the Development of UIs for Mobile Computers," *Proc. Conf. Intelligent User Interfaces*, pp. 69-76, 2001.
- [11] IBM WebSphere Transcoding Publisher, <http://www.ibm.com/software/webservers/transcoding/>, 2003.
- [12] G. Menkhaus and S. Fischmeister, "Evaluation of User Interface Transcoding Systems," *Proc. Seventh World Multiconf. Systemics, Cybernetics and Informatics*, 2003.
- [13] G. Mori, F. Paternò, and C. Santoro, "CTTE: Support for Developing and Analysing Task Models for Interactive System Design," *IEEE Trans. Software Eng.*, vol. 28, no. 8, pp. 797-813, Aug. 2002.
- [14] G. Mori, F. Paternò, and C. Santoro, "Tool Support for Designing Nomadic Applications," *Proc. Conf. Intelligent User Interfaces (IUI '03)*, 2003.
- [15] K. Mullet and D. Sano, *Designing Visual Interfaces*. Prentice Hall, 1995.
- [16] B. Myers, S. Hudson, and R. Pausch, "Past, Present, Future of User Interface Tools," *ACM Trans. Computer-Human Interaction*, vol. 7, no. 1, pp. 3-28, Mar. 2000.
- [17] J. Nichols, B.A. Myers, M. Higgins, J. Hughes, T.K. Harris, R. Rosenfeld, and M. Pignol, "Generating Remote Control Interfaces for Complex Appliances," *Proc. ACM Symp. User Interface Software and Technology*, pp. 161-170, 2002.
- [18] D. Olsen, S. Nielsen, and D. Parslow, "Join and Capture: a Model for Nomadic Interaction," *Proc. ACM Symp. User Interface Software and Technology*, pp. 131-140, 2001.
- [19] L. Paganelli and F. Paternò, "A Tool for Creating Design Models from Web Site Code," *Int'l J. Software Eng. and Knowledge Eng.*, vol. 13, no. 2, pp. 169-189, 2003.
- [20] F. Paternò, *Model-Based Design and Evaluation of Interactive Application*. Springer Verlag, 1999.
- [21] F. Paternò and A. Leonardi, "A Semantics-Based Approach to the Design and Implementation of Interaction Objects," *Computer Graphics Forum*, vol. 13, no. 3, pp. 195-204, 1994.
- [22] F. Paternò and C. Santoro, "One Model, Many Interfaces," *Proc. Fourth Int'l Conf. Computer-Aided Design of User Interfaces*, pp. 143-154, 2002.
- [23] S. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd, "ICrafter: A Service Framework for Ubiquitous Computing Environments," *Proc. Int'l Symp. Ubiquitous Computing*, pp. 56-75, 2001.
- [24] A. Puerta and J. Eisenstein, "Towards a General Computational Framework for Model-Based Interface Development Systems," *Proc. ACM Conf. Intelligent User Interfaces*, pp. 171-178, 1999.
- [25] A. Puerta and J. Eisenstein, "XIML: A Common Representation for Interaction Data," *Proc. ACM Conf. Intelligent User Interfaces (IUI '02)*, pp. 214-215, 2002.
- [26] C. Rich and C. Sidner, "COLLAGEN: A Collaboration Manager for Software Interface Agents," *User Modeling and User-Adapted Interaction*, vol. 8, nos. 3/4, pp. 315-350, 1998.
- [27] P. Szekely, P. Sukaviria, O. Castells, J. Muthukumarasamy, and E. Salcher, "Declarative Interface Models for User Interface Construction Tools: The MASTERMIND Approach," *Eng. for Human-Computer Interaction*, L.J. Bass and C. Unger, eds., pp. 120-150, 1995.
- [28] XForms-The Next Generation of Web Forms, <http://www.w3.org/MarkUp/Forms/>, 2003.



Giulio Mori received a degree in informatics engineering from university of Pisa and is research assistant at the Human Interfaces in Information Systems Research Laboratory of ISTI-CNR, working on design and development of interactive applications.



Fabio Paternò is senior researcher at ISTI-CNR, where he leads the Human Interfaces in Information Systems Research Laboratory. He has been the coordinator of a number of European and other types of projects on user interface software-related topics. He is a member of the IFIP TC13 technical committee. He was the president of ACM-SIGCHI, Italy, from 2000 to 2004. He has been member of the program committees of the main international

HCI conferences. He has published more than 120 papers in refereed international conferences or journals. His main interests are in methods and tools for design and evaluation of usable interactive software systems accessible through many types of contexts.



Carmen Santoro received a degree in computer science from the University of Pisa and since 1997 she has been working as a researcher in the Human Interfaces in Information Systems Laboratory of ISTI-CNR. She has published papers in international conferences and journals. She has been member of the program committee of international conferences like Mobile HCI 2002, INTERACT 2003, and HESSD 2004, and reviewer for international HCI conferences, journals, and books. She was the workshop and tutorial chair of Mobile HCI 2002. Her current research interests include methods and tools for multimodal and multiplatform user interface design and evaluation.

► For more information on this or any computing topic, please visit our Digital Library at www.computer.org/publications/dlib.