TERESA: An Environment for the Design and Development of Multi-Platform User Interfaces

G.Mori, F.Paternò, C.Santoro
ISTI-CNR
56100 Pisa, Italy

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 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 preserving usability. The transformations are supported by an authoring tool, TERESA, which provides designers and developers with various levels of automatic support and possibilities regarding how to tailor such transformations to their needs.

Index Terms - D.2.2 Design Tools and Techniques, H.5.2 User Interfaces, Heterogeneous clients. Multi-platform 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. We mean for platform a class of devices that share the same characteristics, especially in terms of interaction resources. They range from small devices such as interactive watches to very large flat displays. Examples of platforms are the desktop, the PDA, the mobile phone.

The availability of such platforms has forced designers to strive to make applications run on a wide spectrum of platforms 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. If, 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 satisfaction of the users 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 give rise to the fundamental issue of how to assist software designers and developers in building such applications, with the consequence that there is a need for novel methods and tools for the development of interactive software systems able to adapt to different targets while preserving usability.

In current practise the design of multi-platform applications is often obtained through the development of several versions of the same applications, one for each platform considered. Then, such versions can at most exchange data. This solution with no tool support to address multi-platform 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. An example is HTML to WML transcoding

[I02]. However, this type of solution often provides bad results in terms of usability because it assumes that the same tasks are supported by each platform and tends to support them in the same manner without taking into account the specific features of the platform at hand.

Another solution proposed is the use of style sheets. Each platform is associated with a different set of stylesheets. 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, not to mention the fact that even mutual relations among tasks performed through several platforms may exist.

A more comprehensive solution can be obtained through the use of model-based approaches, different levels of abstractions can capture relevant information without having to deal with a plethora of details related to each device and allow a tool to generate the specific version adapted for each device, thus representing a viable alternative to overcome the limitations of other approaches.

In the paper we first describe the basic concepts characterising our approach. Then we move on to describe how we have modified a previously developed tool for task modelling in order to support this new approach. The second part of the paper is devoted to describing the tool (TERESA) that has been designed and implemented to support this approach, in particular we focus on the transformations that it supports, and the XML languages describing the various levels of abstraction considered. Lastly, some concluding remarks along with indications for future work are provided.

2. Design of Multi-platform applications

2.1 Basic concepts

Our approach can be summarised in four words: *One Model, Many Interfaces* [PS02]. 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 (see Figure 1). 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. Then, we allow designers to obtain effective user interfaces for the various platforms considered through a number of transformations implemented by our new tool TERESA (Transformation Environment for inteRactivE Systems representAtions).

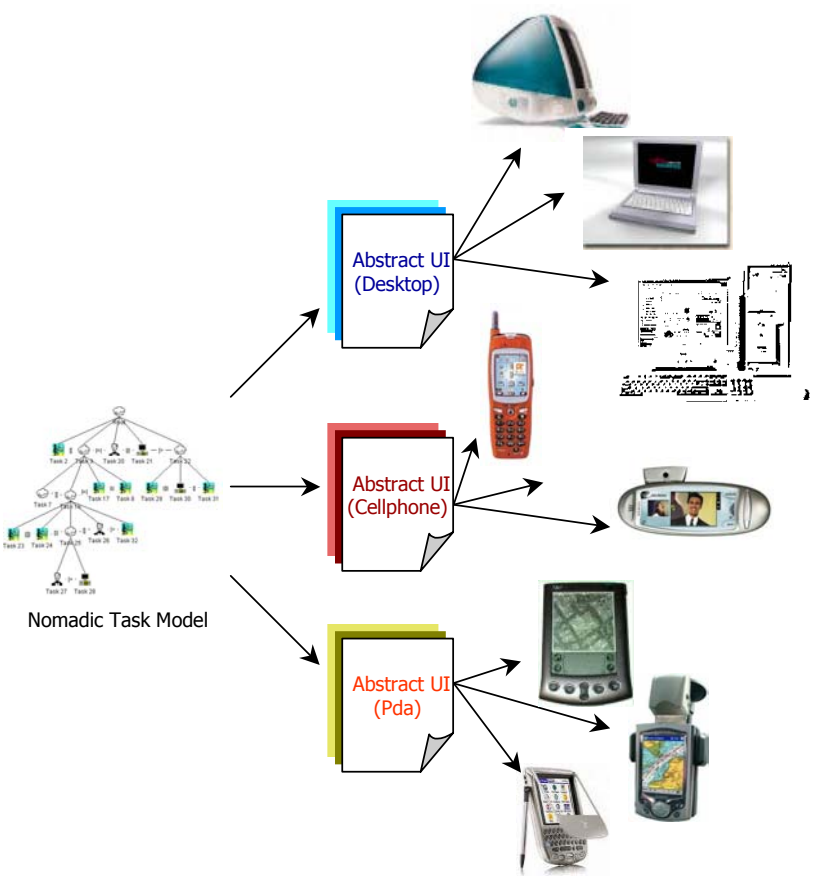

Figure 1: The *One Model, Many Interfaces* approach.

In order to address such issues it is important to consider the various abstraction 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. Only the main aspects are considered, thereby avoiding low level details. An abstract user interface is defined in terms of presentations, identifying the set of user interface elements perceivable at the same time, and each presentation is composed of a number of interactors [PL94], which are abstract interaction objects identified in terms of their main semantics effects.
- *Concrete user interface*, at this point each abstract 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 behaviour.
- *Final User interface*, at this level the concrete interface is translated into an interface defined by a specific software environment (e.g. XHTML, Java, …).

To better understand such abstraction 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 needed to support such tasks and the types of effects that they have to support. For example, for specifying departure and arrival towns we need

selection interaction objects. When we move on to the concrete user interface, we need to consider the specific interaction objects supported. 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 type and size of the font, the foreground and background colours, and decoration images and will be implemented in a specific language.

An early version of our approach and tool was introduced in [MPS03], in this paper we are able to present an engineered solution, supported through a number of XML-based representations and transformations with various levels of automation. We also present the associated tool.

2.2 Tasks and multi-platform 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 instance, on a mobile system users can effectively check the status of a particular flight or get road directions while

driving. On the other hand, while interacting through a desktop system users wish to perform tasks such as gathering information on a company including graphical representations of geographical information, or reading a movie review while its trailer is displayed.

- *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 available.
  - *With different user interface objects*, the same task can be supported through different interaction objects according to the media and the resources available. Figure 2 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 little screen is available (right part).
  - *With different task decomposition*, when a main task needs to be accomplished it can be structured differently. 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 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 particular 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.

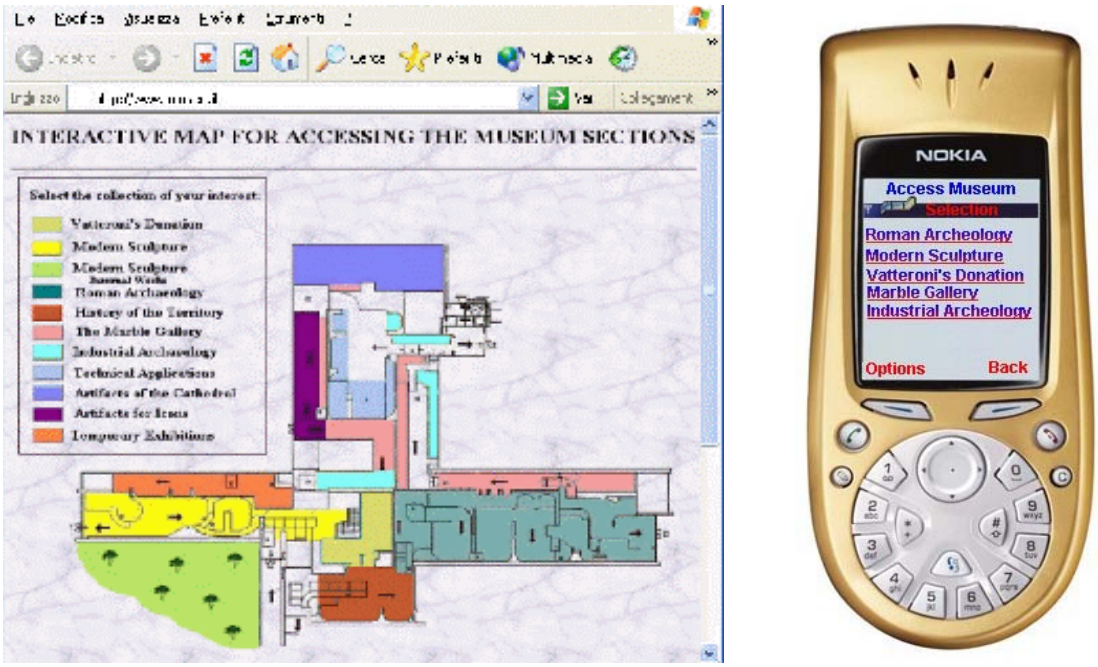

Figure 2: Example of different interfaces supporting the same task through different platforms.

2.3 Related Work

The most common model-based approach in software engineering, UML [BRJ99], 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 support development of desktop interactive applications, examples are Adept [WJKCM93], Mobi-D [PE99], Mastermind [SSC95] or how to use such models to support user interaction at run-time, still in desktop applications [RS98]. As Myers and others [MHP00] 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 all of these into account. Even recent W3C standards, such as XForms [W3C], 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 controls suitable for general-purpose use. So, to some extent it applies concepts that have long been discussed in the research area concerning model-based design of human-computer interaction. The types of abstractions supported by XForms are at the level of an abstract user interface. 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.

Calvary, Coutaz and Thevenin have defined the concept of plasticity to indicate user interfaces able to adapt to different platforms while preserving usability [CCN97]: TERESA is a tool for the design and development of applications supporting such concept. The new challenges have raised the need to define XML-based languages for representing the relevant

concepts and ease their automatic manipulation. Examples of projects that have addressed these issues are: *The User Interface Markup Language (UIML)* (http://www.uiml.org/) [APB99] is an XML-compliant language that allows a declarative description of a user interface in a device-independent manner. This has been developed mainly by Harmonia and Virginia Tech. However, their tools do not support the task level. *The Abstract User Interface Markup Language (AUIML)* is an XML vocabulary designed to allow definition of the intention of an interaction. This work has mainly been carried out at IBM and has quite limited tool support. *The eXtensible Interface Markup Language (XIML)* (http://www.ximl.org/) [PE01] is an extensible XML 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. Tool support is not currently available. While these approaches have shown some interesting results, there is still a lack of general solutions able to support the various relevant abstraction levels.

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 [BV02] and WebRevEnge [PP03]: 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 multi-platform 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. Next sections describe in detail each step.

We start with *High-level task modelling of a multi-platform 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.

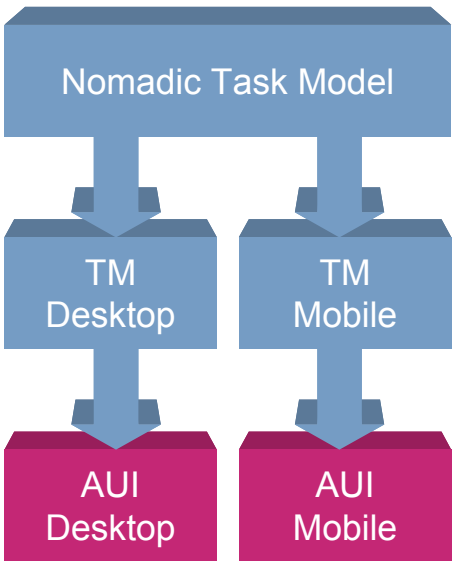

Figure 3: Example of design process for a nomadic application.

The next step 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 task model, depending on the specific platform considered thus obtaining the various platform-dependent task models, which represent the input of the next step. Figure 3 shows an example of the resulting process in the case of a nomadic application that can be accessed through the desktop and the mobile platform.

Then designers have to move *from the system task model to the abstract user interface*. 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. The presentation part 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 which should be grouped together because they are closely related to each other). In order to support such transformations, we have defined an XML format for the task model language and for the abstract user interface language.

The next step 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 operators also have to be appropriately implemented by 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 colour.

The last phase 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.

In order to provide tool support for this method we have extended a tool for task modelling that we developed beforehand and we have developed from scratch the TERESA tool for authoring multi-platform applications. The functionality of these two tools will be described in the next sections.

4. Task Models of Nomadic Applications

The ConcurTaskTrees Environment (CTTE) [MPS02] is a highly engineered, publicly available (http://giove.cnuce.cnr.it/ctte.html) tool for task modelling. It eases the development of task models described using the ConcurTaskTrees (CTT) notation [P99] and supports their analysis through a number of features (metrics evaluation, interactive simulation, …). 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, 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) and a number of attributes (such as frequency) (see Figure 4).

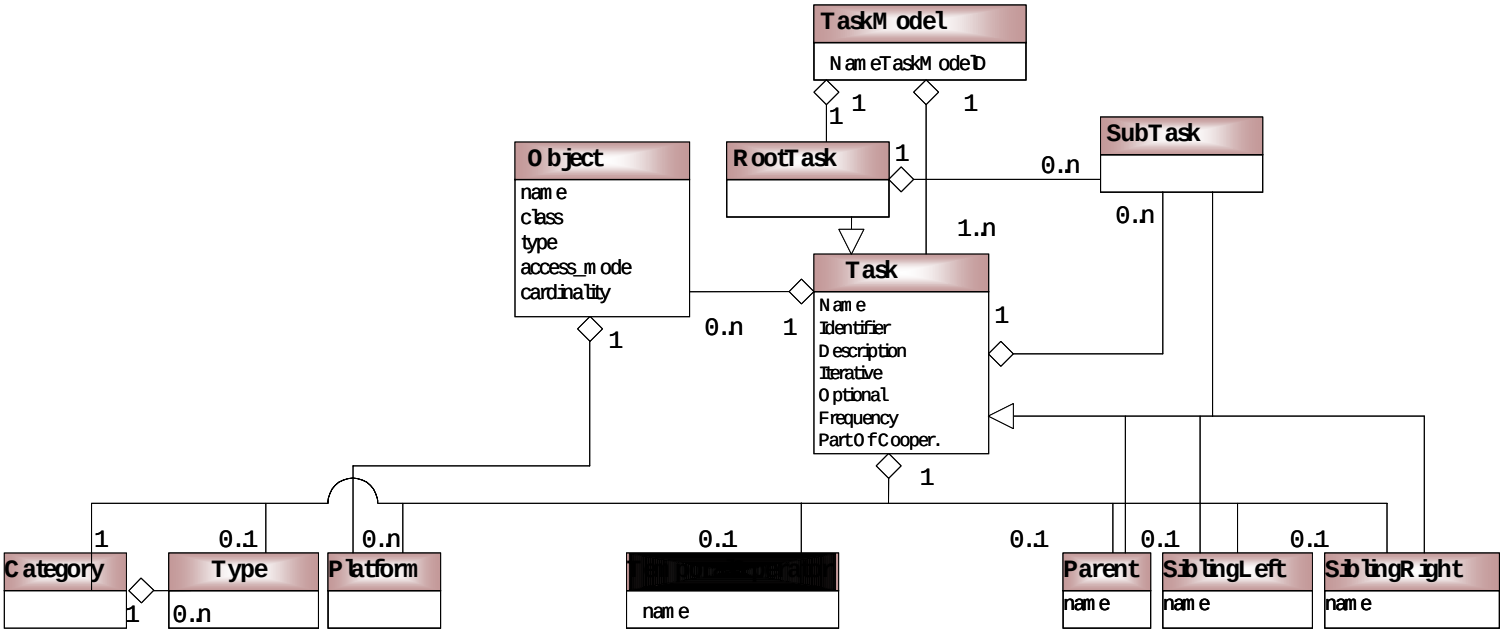

Figure 4: Description of the concepts represented in the ConcurTaskTrees notation for task models.

In order to support this approach we needed to extend CTTE in such a way to be able to capture the specific aspects of task models for nomadic applications:

- The platform attribute has been added in each task specification; its purpose is to indicate the types of platforms that are suitable to support it. As far as this attribute is concerned, it is worth noting that at this level –the task level- *classes* of devices sharing certain similarities are considered, rather than *specific* devices. So, in our framework we made provision for typical sample device clusters as mobile phones and PDAs are, together with the possibility for the users to define their own device classes. This proved to be both feasible and flexible solution to face up to the problem of dealing with the disparate platforms that our approach had to consider. Nevertheless, additional levels of refinement within the same cluster are considered in the last phase of the method, when the knowledge of more specific characteristics of the devices considered becomes useful for producing effective final user interfaces. Such characteristics might range from features

like screen size and network capacity to information related to the specific operating system available, together with multimedia and/or multimodal capabilities of the devices.

- 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 effectively 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 the global picture of the application, which, if appropriately reflected and conveyed in the user interfaces will have a beneficial effect also on the final users.

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 and others pointed out [MHP00] 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*, XML-based languages have been proposed for every type of domain. In the field of interactive systems there have been a few proposals that partially capture the key aspects to be addressed.
- *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 this purpose, we have also developed a tool (WebRevEnge [PP03]) that allows building task models from web site code. It complements the approach pursued 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, …. This simply requires extending the implementation of the

last transformation (from the concrete to the final user interface) for the specific software environments.

5.2 Main functionality

TERESA is a transformation-based environment 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 set of tasks which are enabled over the same period of time according to the constraints indicated in the model (*Enabled Task Sets*). Such sets, depending on the designer's application of a number of heuristics supported by the tool, are grouped into a number of *presentation task sets* and related *transitions*. Since there is a direct correspondence between the specific platform-dependent task model and the enabled task sets identified accordingly, it is evident that the selection of the specific platform considered has an impact on the identification of such sets and in turn on the overall organisation of the final user interface.
- *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 interface, in terms of both its static structure (the "presentation" part) and dynamic behaviour (the "dialogue" part), is saved for further analyses and transformations. It is worth pointing out that 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 results in a significant improvement 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.* Through this option the tool automatically generates the final UI for the target platform, starting with the currently visualised single-platform task model, and using a number of default configuration settings related to the user interface generation.

5.3 TERESA Abstract User Interface language

An abstract user interface is composed of a number of presentations, which 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.

The structure of the presentation is defined in terms of interactors (also called Abstract Interaction Objects or AIO) and their composition operators (see Figure 5). It is possible to distinguish between interactors supporting user interaction (interaction_aio) and those that present results of application processing (application_aio).

Each interaction_aio defines an abstract interaction object which implies an interaction between the user and the application. Such an interaction can be of different types depending on the type of task supported. We have selection_aio (to select between a set of elements),

edit_aio (to edit an object), control_aio (to trigger an event within the user interface). Differently from an interaction_aio, an application_aio defines an abstract application object which implies an action only from the application. Each application_aio can be of different types (text_aio, object_aio, description_aio, feedback_aio) 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.

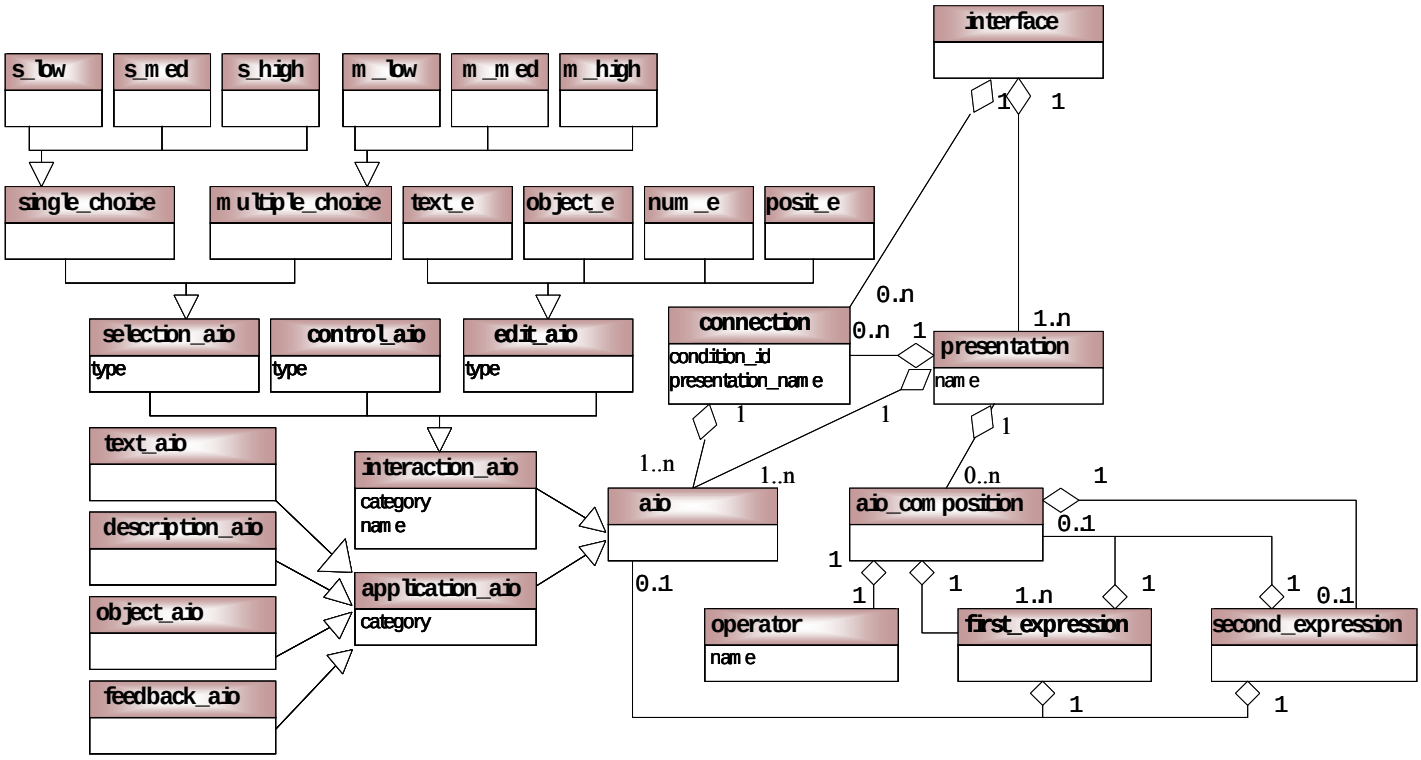

Figure 5: The concepts and their relations represented in the TERESA notation for abstract user interfaces.

The composition operators can involve one or two expressions, each of them can be composed of one, 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 [MS95]. 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 Enabled Task Sets

The Enabled task sets (ETSs) are sets derived from a CTT task model, and represent tasks that are enabled over the same period of time. In particular, the ETSs are derived from analysing the formal semantics of the CTT temporal operators: for example, if two tasks are concurrently executed, they are enabled at the same time, so they belong to the same enabled task set. Alternatively, if two tasks are connected through an enabling operator, the second task will be executed just after the execution of the first one, so the tasks do not belong to the same enabled task set.

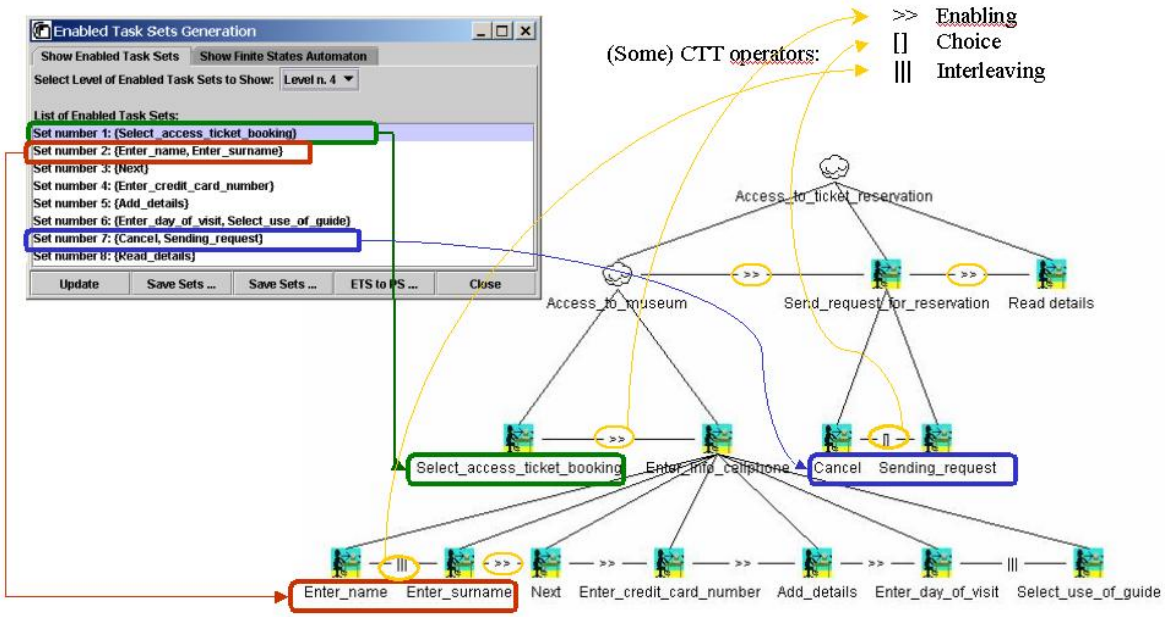

Figure 6: Example of calculation of enabled task sets.

Figure 6 shows an example of a set of enabled task sets associated with a simple task model related to the *Access to ticket reservation* task. 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 set composed of only itself. Then, there are two subtasks (*Enter name* and *Enter surname*) composed using the interleaving operator (|||) and followed by another enabling operator (the interleaving operator has higher priority than the enabling one). This means that they are enabled over the same period of time and, only after the performance of both of them, the other tasks are enabled. So, there is another enabled task set associated with them. In an another part of the model there is an instance of the choice operator ([]) between the *Cancel* and the *Sending request* subtasks. Since the user should be able to choose between one of them, they are enabled over the same period of time and belong to the same enabled task set.

It is worth pointing out that the task model shown in Figure 6 describes the activity of reserving a ticket provided that a specific platform -the mobile phone in this case- is supposed to be 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 might be executed concurrently with the task of entering credit card number). Hence, another task model is specified accordingly, which in turn leads to a modified arrangement of ETS.

The calculation of the enabled task sets implies the calculation of the conditions that allow passing from ETS to ETS, which depends on the temporal operators existing among the various tasks. Such conditions can be presented in various manners. One example is when it is just a task that allows 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 concurrently performed and such a group is connected with an enabling operator to another set. In this case, whatever the execution order of those tasks, the condition for moving to the next set of tasks is the fact that only when all the tasks are carried out, the next set is enabled. As a result, the Boolean condition expressing such a constraint is an AND operator applied to all the involved tasks. 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 the enabled task sets generated is of the same order as the number of the CTT enabling operators appearing in the task model, a direct mapping between the enabled task sets 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 in obtaining a lower number of presentations. In particular, once the ETSs have been defined, we need to specify some rules to reduce their number by merging two or more ETSs into new sets, called *Presentation Task Sets* or PTSs. The reasons for such step are various: first of all, reducing the initial number of ETSs which -as we previously note-d in some cases can be very high; secondly, keeping and highlighting in the same presentation significant information (as a data exchange is) even when the involved tasks belong to different ETSs so that users can better follow the flow of information; lastly, avoiding repeatedly considering groups of tasks that all

share a similar structure. 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) ETSs differ for only one element, and those elements are at the same level connected with an enabling operator, they can be joined together.
- If an ETS is composed of just one element, it should be joined with another ETS that shares some semantic feature.
- If some ETSs 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 in only one 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 amongst the various sets, the next step is obtaining the description of the abstract user interface in terms of abstract user interface objects 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.

6.2.1 Transforming tasks into abstract interaction objects

In order to map the various tasks into the related abstract user interface objects, 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 abstract user interface object (respectively, interaction_aio or application_aio). Each of these interactor categories represents a class 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 specialised subclasses up to reaching elementary interactors. The last step for deciding the type of interactor analyses the semantics effects of the interactions to support. A number of interactors have been identified (numerical, text, object, position) for this purpose. Then, 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 addition, particular task types allow the specification of attributes which are peculiar. For selection tasks, besides specifying the type of selection supported (single or multiple), it is also possible defining the cardinality of the set of objects which the selection will be

performed from (high/medium/low). For these tasks, the information about the cardinality of the set is useful to select the appropriate type of interactor able to support the selection.

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. Also in this case there are interactors suitable to support these activities.

6.2.2 Identification of the interactor composition operators

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

In fact, on the one hand, there are some CTT operators which are directly linked with operators of the abstract user interface. It is the case when two or more tasks sequentially performed end up in the same presentation task set: the sequentiality 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 amongst them. For example, it is the case of the Hierarchy operator, whose 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.

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 system 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 widgets appearing on the user interface (menus, buttons, images, labels, etc.), whereas the "application" objects have a more abstract connotation as they 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 an attribute 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 needs to communicate with the interactor handling the perceivable object associated with the task.

7. From the abstract user interface to the concrete user interface

One of the main goals in the design of TERESA is 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. At the beginning a number of general parameters and information are presented: as you can see in Figure 7, 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 to further select one of these options and visualise/modify 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 (see Figure 7). 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, the hierarchy operator through different font sizes, the ordering by means of an ordered list, and the relation operator by means of a form. Once a designer selects a specific type of communication technique, its preview is highlighted accordingly besides the pull-down menu in order to facilitate the understanding of the main characteristics of the resulting design.

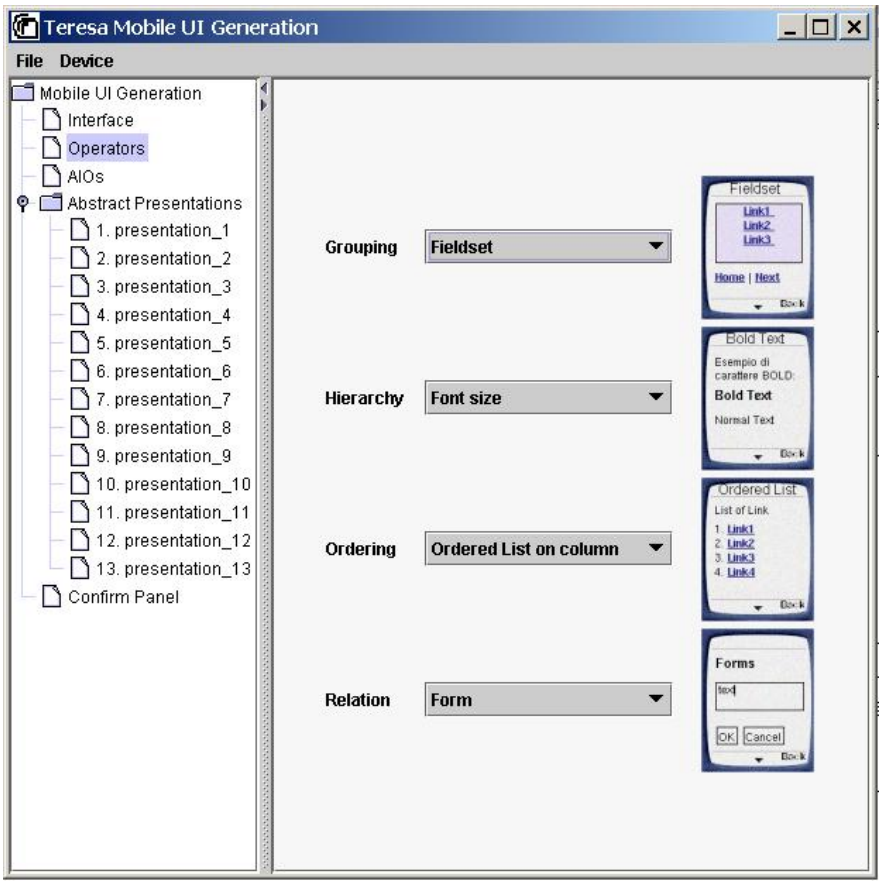

Figure 7: Panel for selecting the implementation of each composition operator in the abstract language.

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 could be implemented with different techniques depending on whether the desktop or the mobile platform is considered. In fact, on the one hand the desktop environment allows the use of 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 colours). On the other hand, the small capability of a mobile phone does not allow implementing the grouping operator by using an unordered list of elements by column then this technique is not available on this platform.

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 too much the dimension of the text without compromising the quality of the result.
In addition, other differences can be found in supporting the user interface design between the two 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 some cellphone systems they cannot be supported due to the limited capabilities of the considered platform.

In the prototyping phase the designer can select any presentation and change either how to implement a composition operator in it (see an example of grouping specification in Figure 8) or a specific interaction object 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 and are spared starting the process from scratch every time.

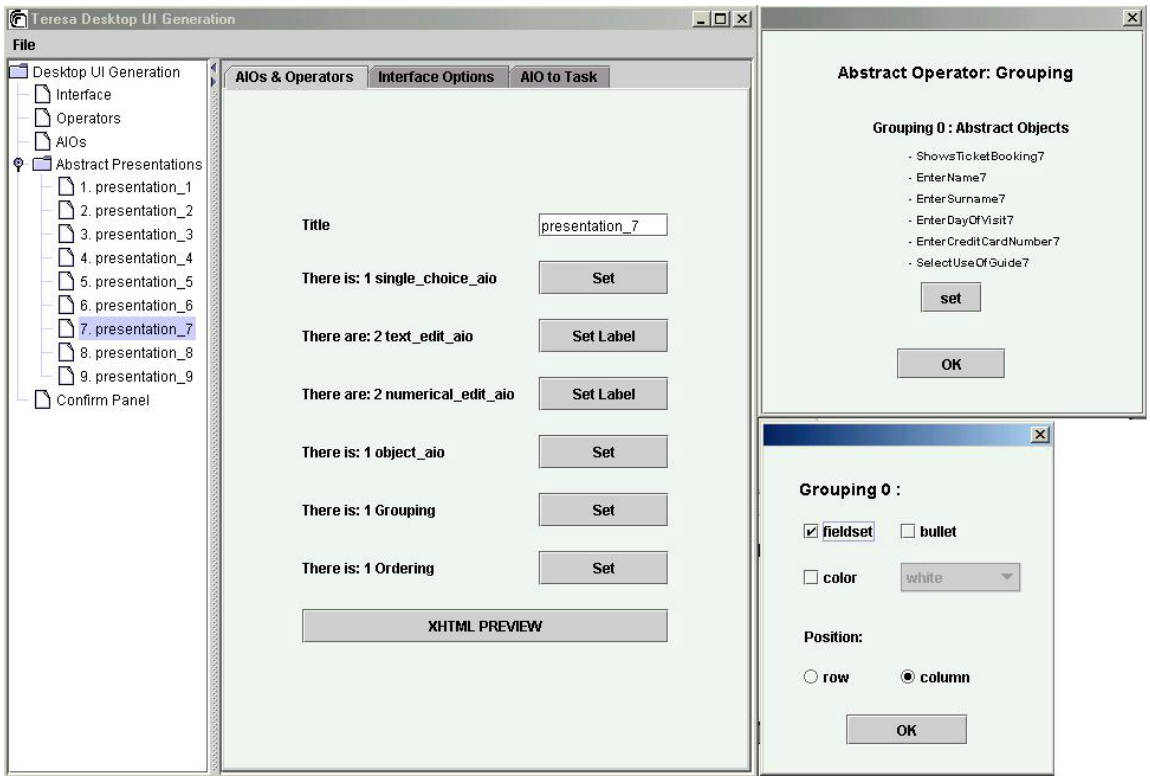

Figure 8: Possibility of defining parameters in a specific presentation.

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, 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.

Figure 9 shows how the result of the prototyping process is shown to the designers: an overall summary table is provided by the tool in order to allow designers to understand the design criteria currently applied. As you can note, in presentation 3 there are two grouping operators which have been currently implemented with two different techniques (fieldset and unordered list on columns), as a result of a further designer's customisation of presentation 3.

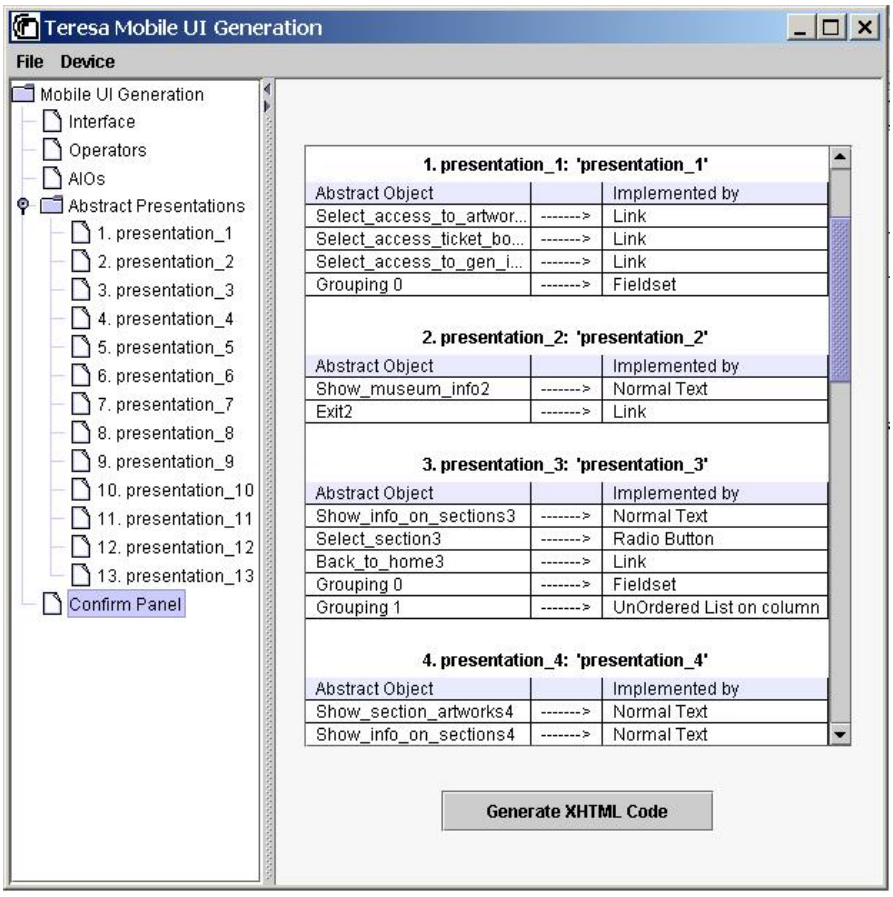

Figure 9: Panel summarising the design choices.

8. EXPERIENCES of USE and LESSONS LEARNT

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 modelling students were introduced to the tool for developing and analysing 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 multi-platform application. This demonstrates 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 experiment was conducted at the Motorola Italy software development centre [CFMRB03] where five developers were asked to participate in a 30 minute preparation session and to dedicate 10 minutes reading the TERESA help prior to start the exercises. The first experiment consisted in starting with a given task model and obtaining the corresponding user interface for both desktop and mobile phone. A second experiment was conducted in order to collect more information about satisfaction and cost/effectiveness of the approach. Results showed similar total times both for the TERESA approach and the traditional one, with opposite impact on different phases. The use of the tool almost doubled required time at design stage, while at development stage the results show a dramatically improved prototyping performance, reducing required 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 slight total time increase is acceptable since it involves a trade-off with design overall quality: many subjects appreciated the benefits of a formal process and support to individuate 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 more consistent look of the pages resulting from the model-based approach, as well as the reduced risk of forgetting the formal specifications. This is also coupled with an increased consistence between desktop and 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.

8. CONCLUSIONS

We have presented a method and the associated tool supporting design and development of nomadic applications. It allows designers to use the results of a conceptual analysis in terms of tasks and their relations as input for supporting the user interface prototyping, taking into account the characteristics of the platform considered. Designers have different levels of control over the development process. The resulting environment supports heterogeneous interaction devices through multiple levels: at the task level it is possible to identify what activities can be effectively supported by each platform, at the abstract user interface level it is possible to define what interactions should be supported and then, with the support of the tool, it is possible to derive a corresponding usable interface. The tool is able to store the mappings among the various abstraction levels. This information can be useful for designers for various purposes. The tool is publicly available at http://giove.cnuce.cnr.it/teresa.html

This approach has been applied to the development of two nomadic applications: one in the museum application domain and an office application.

While the current TERESA version supports the design and development for graphical and vocal interfaces for various platforms (currently through the generation of XHTML, XHTML Mobile Profile and VoiceXML [BP03] but other software environments will be considered soon), further work will be dedicated to support a broader set of modalities and their combinations.

Acknowledgments

We gratefully acknowledge support from the European Commission through the CAMELEON IST project (http://giove.cnuce.cnr.it/cameleon.html). We also thank Francesco

Correani for the help in the implementation of the abstract-to-concrete user interface transformation.

References

[APB99] 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. Available at http://www.harmonia.com/resources/papers/www8_0599/index.htm

[BP03] Berti S., Paternò F., Model-based Design of Speech Interfaces, Proceedings DSV-IS 2003, Madeira, June 2003, Springer Verlag.

[BV02] Bouillon, L., Vanderdonckt, J., Retargeting Web Pages to other Computing Platforms, Proceedings of IEEE 9th Working Conference on Reverse Engineering WCRE'2002 (Richmond, 29 October-1 November 2002), IEEE Computer Society Press, Los Alamitos, 2002, pp. 339-348.

[BRJ99] Booch, G., Rumbaugh, J., Jacobson, I., *Unified Modeling Language Reference Manual,* Addison Wesley, 1999

[CCN97] Calvary, G., Coutaz, J., Thevenin, D., A Unifying Reference Framework for the Development of Plastic User Interfaces, Proceedings Engineering Human-Computer Interaction, pp.173-192, 2001.

[CFMRB03] Chesta, C., Fliri, M., Martini, S., Russillo, B., Barbero, C., First Evaluation of Tools and Methods, CAMELEON Project Document: D 3.4, July 2003.

[EVP01] Einsenstein, J., Vanderdonckt, J., Puerta, A. *Applying Model-Based Techniques to the Development of UIs for Mobile Computers*, Proceedings IUI'01: International Conference on Intelligent User Interfaces, pp 69-76, ACM Press, 2001.

[I02] IBM WebSphere Transcoding Publisher, http://www.ibm.com/software/webservers/transcoding/

[MHP00] Myers, B., Hudson, S., Pausch, R. *Past, Present, Future of User Interface Tools*. Transactions on Computer-Human Interaction, ACM, 7(1), March 2000, pp. 3-28.

[MPS03] Mori, G., Paternò, F., Santoro, C., "Tool Support for Designing Nomadic Applications", Proceedings ACM IUI'03, Miami, ACM Press.

[MPS02] Mori, G., Paternò, F., Santoro, C., *CTTE: Support for Developing and Analysing Task Models for Interactive System Design*, IEEE Transactions on Software Engineering, pp.797-813, 2002, Vol. 28, N. 8.

[MS95] Mullet, K., Sano, D., *Designing Visual Interfaces*. Prentice Hall, 1995.

[P99] Paternò, F., Model-Based *Design and Evaluation of Interactive Application*. Springer Verlag, ISBN 1-85233-155-0, 1999.

[PL94] 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.

[PP03] Paganelli, L. Paternò, F. A Tool for Creating Design Models from Web Site Code, International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing 13(2), pp. 169-189 (2003).

[PS02] Paternò, F., Santoro, C., *One Model, Many Interfaces*, Proceedings Fourth International Conference on Computer-Aided Design of User Interfaces, pp. 143-154, Kluwer Academics Publishers, Valenciennes, 2002.

[PE99] Puerta, A., Eisenstein, J., *Towards a General Computational Framework for Model-based Interface Development Systems*, *Proceedings ACM IUI'99*, pp.171-178.

[PE01] Puerta, A., Eisenstein, *XIML: A Common Representation for Interaction Data*, *Proceedings ACM IUI'01*, pp.214-215.

[RS98] Rich, C., Sidner C., *COLLAGEN: A collaboration manager for software interface agents*, User Modelling and User-Apadted Interaction, 1998, 8(3/4), pp.315-350.

[SSC95] Szekely, P., Sukaviria, P., Castells, O., Muthukumarasamy, J., Salcher, E., *Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach*. In *Engineering for Human-Computer Interactio*n, L.J. Bass and C. Unger (eds), Chapman & Hall, London, 1995, pp 120-150.

[W3C] XForms – The Next Generation of Web Forms, http://www.w3.org/MarkUp/Forms/

[WJKCM93] Wilson, S., Johnson, P., Kelly, C., Cunningham, J. and Markopoulos, P., "Beyond Hacking: A Model-based Approach to User Interface Design". *Proceedings HCI'93*. pp.40-48, Cambridge Univ. Press, 1993.