Tool Support for Designing Nomadic Applications

Giulio Mori, Fabio PaternÚ, Carmen Santoro
I.S.T.I. ñ C.N.R.
Via G. Moruzzi, 1
56100 Pisa, Italy
+39 050 315 3066
{g.mori, f.paterno, c.santoro}@cnuce.cnr.it

ABSTRACT
Model-based approaches can be useful when designing nomadic applications, which can be accessed through multiple interaction platforms. Various models and levels of abstraction can be considered in such approaches. The lack of automatic tool support has been the main limitation to their use. We present a tool, TERESA, supporting top-down transformations from task models to abstract user interfaces and then to user interfaces for different types of interaction platforms (such as mobile phones or desktop systems). It allows designers to keep a unitary view of the design of a given nomadic application. Moreover, the tool provides support for obtaining effective interfaces for each type of platform available, taking into account the consequent differences in terms of tasks and their performance.

Categories and Subject Descriptors
D.1.7 [**Visual Programming**]; D.2.2 [**Design Tools and Techniques**]: User interfaces; H.5.2 [**User Interfaces**]: User interface management systems (UIMS); I.2.4 [**Knowledge Representation Formalisms and Methods**].

General Terms
Design, Human Factors, Languages.

Keywords
Multi-platform applications, Model-based design, Tool support for designers.

1. INTRODUCTION
Designing applications that exploit new multi-platform technology is often a difficult problem. For software developers this introduces the problem of constructing multiple versions of single applications and endowing these versions with the ability to respond to changes in context. Creating different versions of applications for different devices engenders extra development and expensive maintenance costs of cross-platform consistency and complicates the problems of configuration management.

In a recent paper, discussing the future of user interface tools, Myers, Hudson, and Pausch [3] indicate that the wide platform variability encourages a return to the study of some techniques for device-independent user interface specification. The results of this approach is that developers can define the input and output needs of their applications, vendors can describe the input and output capabilities of their devices, and users can specify their preferences. Then, the system can choose appropriate interaction techniques taking all of these into account.

The basic idea is that, instead of having separate applications for each device that exchange only basic data, there is some abstract description and then an environment that is able to suggest a design for a specific device geared towards its features and possible contexts of use. This is the main goal of the model-based approaches [6][8][11] that have been considered, though not extensively accepted in the last decade.

However, nomadic applications raise new challenges that can be better addressed using a model-based approach. There is a need for a unitary view of nomadic applications, even if their parts require different instantiation for different platforms. This allows designers to understand and control the dependencies among such instances. Secondly, new design criteria suitable for mobile devices should be introduced.

In our method [7] we focus on models that can support development of user interfaces while preserving usability, in particular task models specifying the different activities that are supposed to be performed in an interactive system. Such models should be developed involving users so as to represent how they prefer to perform activities.

In order to support development of nomadic applications we have designed and developed the TERESA (Transformation Environment for inteRactivE Systems representAtions) tool providing general solutions that can be tailored to specific cases. This tool supports transformations in a top-down manner, providing the possibility of obtaining more concrete descriptions starting with abstract representations.

In the paper after some discussion of related work, we first introduce the method that we have developed to support design of nomadic applications followed by a discussion of the relations among tasks and platforms and how we had to improve the CTTE tool in order to allow designers to better capture such flexible relations. Then, we move on to provide an overall description of the first version of the TERESA tool. We devote a good deal of attention to how the mixed initiative paradigm is supported in TERESA. We also illustrate the XML language used to describe abstract user interfaces. Lastly, some examples followed by concluding remarks are provided.

2. RELATED WORK

The basic idea of how to cope with the current situation of heterogeneity of currently available devices and the need for usable UIs is that, instead of having separate applications for each device, that exchange only basic data, there is some abstract description and an environment able to suggest a design suitable for a specific device.

This problem is a novel challenge for model-based design and development of interactive applications. The potentialities of these approaches have only begun to be addressed. In the GUITARE Esprit project (http://giove.cnuce.cnr.it/guitare.html) a user interface generator was developed: it takes ConcurTaskTrees (CTT) task models [6] and produces user interfaces for ERP applications according to company guidelines. However, automatic generation is not a general solution because of many, varying factors that have to be taken into account within the design process. Semi-automatic support is more general and flexible: Mobi-D [8] is an example of a semi-automatic approach, but it only supports design of traditional graphical desktop applications.

UIML [1] is an appliance-independent XML user interface language. While this language is ostensibly independent of the specific device and medium used for the presentation, it does not take into account the research work carried out over the last decade on model-based approaches for user interfaces: for example, the language provides no notion of task, it mainly aims to define an abstract structure. The W3C consortium has recently delivered the first version of a new standard (XForms) that presents a description of the architecture, concepts, processing model, and terminology underlying the next generation of Web forms, based on the separation between the purpose and the presentation of a form. If it shows the importance of separating conceptual design from concrete presentation, it also highlights the need for meaningful models to support such approaches.

XIML [9] (eXtensible Interface Markup Language, http://www.ximl.org) is a XML-based language, whose initial development took place at the research laboratories of RedWhale Software. It is intended to be a universal user interface specification language, since it provides a way to completely describe a user interface and represent attributes and relations of the important elements of a user interface without worrying about how they will be implemented. In other words, it enables a framework for the definition and interrelation of interaction data items, thereby providing a standard mechanism for applications and tools to interchange interaction data and interoperate within integrated user-interface engineering processes, from design, to operation, to evaluation. Today XIML is probably the most advanced UI specification language, as it can serve for context-sensitivity and many other objectives. However, it is worth noting that XIML mainly focuses on syntactic, rather than semantic aspects. In addition, tool support is not publicly available.

Collagen [10] uses an explicit embedded task model to support the creation of task-aware collaborative agents. The agent interprets and guesses the user's current intentions, and can determine efficient plans to achieve them. The issues related to design of multi-platform applications are not considered in this approach.

ARTStudio [12] is an early research prototype aiming to support design of multi-target applications. However, the possibility of identifying dependencies among tasks performed through different platforms is not provided.

More generally, the issue of applying model-based techniques to the development of UIs for mobile computers has been addressed at a conceptual and research level [2], but there are still many issues that need to be solved to identify systematic, general solutions that can be supported by automatic tools. Our approach aims to support design and development of nomadic applications providing general solutions that can be tailored to specific cases, whereas current practise is still to develop ad hoc solutions with few concepts that can be reused in different contexts.

3. THE METHOD

Our method for model-based design 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:

♦ *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 roles involved, and also a domain model aiming to identify all the objects that have to be manipulated to perform tasks and the relations among such objects. Such models are specified using the ConcurTaskTrees notation, which allows designers to indicate the platforms suitable to support each task.

♦ *Developing the system task model for the different platforms considered*. Here designers have to filter the task model according to the target platform and, if necessary, further refine the task model, depending on the specific device considered, thus, obtaining the system task model for the specific platform.

♦ *From system task model to 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.

♦ *User interface generation*. This phase is completely platform-dependent and has to consider the specific properties of the target device. Then, every interactor is mapped into interaction techniques supported by the particular device configuration considered (operating system, toolkit, etc.), 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 visual interfaces by using presentation patterns such as proximity, similarity and continuity.

4. TASKS AND NOMADIC APPLICATIONS

In our method we focus on models that can support development of user interfaces while preserving usability, in particular, task models specifying the different activities that are supposed to be performed in an interactive system. Such models should be developed involving users so as to represent how they prefer to perform activities. The basic idea is to capture all the relevant requirements at the task level and then be able to use such information to generate effective user interfaces tailored for each type of platform considered. Task models are represented by the CTT notation that supports hierarchically structured models with the possibility of providing a number of temporal and semantic relations and attributes, thus allowing the description of flexible behaviours.

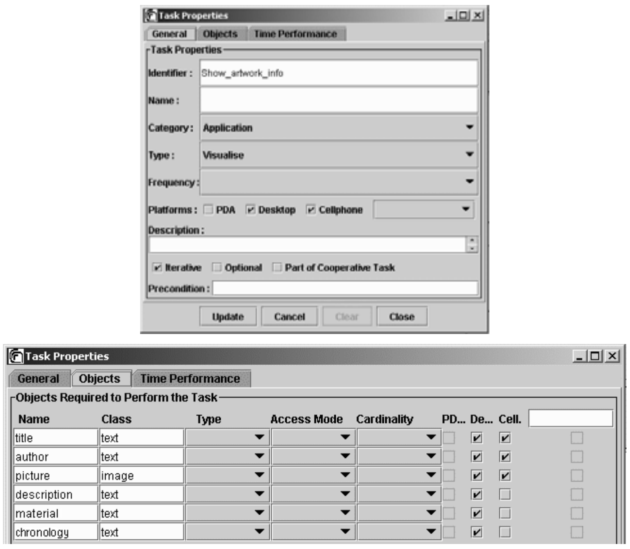
When nomadic applications are considered designers should be careful of the possible relations among tasks and the potential platforms. In some cases the same tasks can be performed in the same manner on different platforms, but often there are other possibilities:

- It is not possible to perform the same task on another platform;
- It is possible to perform the same task on another platform but with different modalities;
- The performance of a task on one platform can enable or disable the performance of another task on another platform.

To cope with these possibilities, we have designed and implemented a new version of CTTE [4] that supports a number of features; one of the most relevant is allowing designers to specify for each task the set of platforms suitable to support it. For example, you can see in Figure 1-(a) that the platforms suitable to support the *Show artwork info* task are desktop and cellphone.

In addition, as far as the platforms are concerned, CTTE allows designers to specify a further level of refinement: since tasks can manipulate a number of objects, it is possible to specify the suitable platforms even at the object level. An example is shown in Figure 1-b: for example the *description* object is supported by just the desktop platform and not by the cellphone, (although the related task was supported by both platforms).

The result of this specification process is an integrated task model where all the tasks supported by the concerned platforms are specified. This is the main input of another feature of the tool that automatically calculates the task model for each of the platforms considered. Figure 2 shows how the CTTE tool supports filtering according to the platform attribute: the designers are given the possibility of selecting one platform from those that have been specified within the whole task model (e.g. the radio button menu items related to platforms other than desktop and cellphone have been automatically disabled).



Task Properties

Identifier : Show_artwork_info

Name :

Category : Application

Type : Visualise

Frequency :

Platforms : ☐ PDA ☑ Desktop ☑ Cellphone

Description :

☑ Iterative ☐ Optional ☐ Part of Cooperative Task

Precondition :

Objects Required to Perform the Task

Name	Class	Type	Access Mode	Cardinality	PD...	De...	Cell.	
title	text				☐	✓	✓	☐
author	text				☐	✓	✓	☐
picture	image				☐	✓	✓	☐
description	text				☐	✓	☐	☐
material	text				☐	✓	☐	☐
chronology	text				☐	✓	☐	☐

Figure 1. The specification of different platforms for each task (a) and for each object manipulated by a task (b) in the CTTEnvironment.

Once the user selects a specific platform, the tool derives the resulting single-platform task model which can be saved separately for further analysis and constitutes one of the inputs for the TERESA tool. The platform-related system task model is obtained by pruning from the model all the tasks that are not relevant for the selected platform. The resulting model may have inconsistencies that can be automatically detected and solved through interaction with the designer.

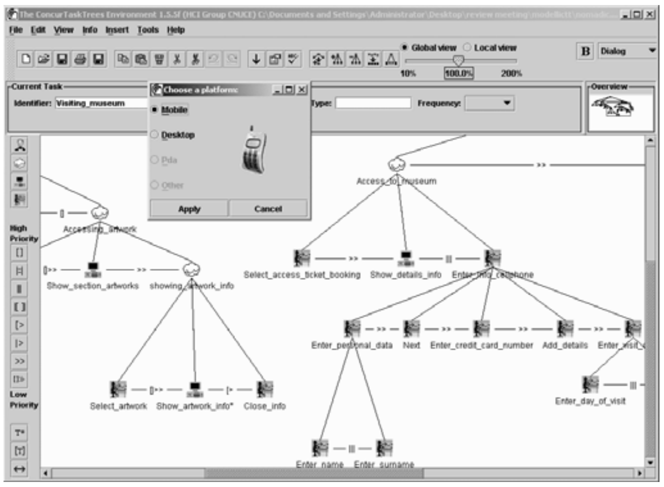


Figure 2. An example of filtering the task model of a nomadic application.

5. TERESA

TERESA is a transformation-based environment that supports the design of an interactive application at different abstraction levels and generate the concrete user interface for various types of platform. In Figure 3 we show the main transformations supported in TERESA, in terms of the XML-based representations that are supported:

- *Presentation sets and transitions generation.* From the XML specification of a CTT task model ("XML CTT Task Model" module in Figure 3) 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 sets* and related *transitions*.
- *From task model -related information to abstract user interface.* Both the XML task model and Presentation Sets specifications are the input for the transformation generating the associated abstract user interface ("XML AUI" module in Figure 3). The specification of the abstract user interface, in terms of both its static structure (the "presentation" part) and dynamic behaviour (the "dialogue" part), is saved for further analyses and transformations.
- *From abstract user interface to concrete interface for the specific platform.* This transformation starts with the loading of an abstract user interface previously saved for a and yields the related concrete user interface for the specific 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, starting with the currently visualised (single-platform) task model, and using a number of default configuration settings related to the user interface generation.

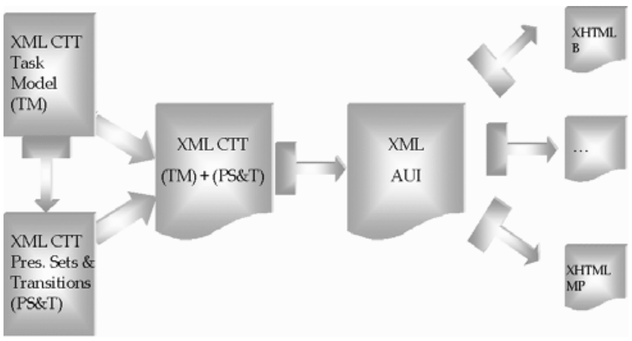


Figure 3. The main transformations in TERESA in terms of XML-based specifications supported.

6. THE TERESA ABSTRACT USER INTERFACE XML LANGUAGE

In this section we provide the general description of the XML TERESA language for abstract user interfaces and then we describe the structure of the language through the most relevant parts of its DTD.

6.1 General Description of the Language

The abstract user interface is mainly defined by a number of presentations defining its static structure, and a number of transitions defining how the user interface evolves over the time. Each presentation is constituted of a set of interactors composed through a number of different operators. For the moment we have identified a number of composition operators that capture typical effects that user interface designers actually aim to achieve [5]:

- ***Grouping (G):*** the idea is to group together two or more elements, so this operator should be applied when the involved tasks share some characteristics. A typical situation is when the tasks have the same parent (they are activities needed to perform a high level task). This is the only operator for which the position of the different operands is irrelevant.
- ***Ordering (O)*** operator: it is applied when some kind of order exists amongst elements. The more intuitive one is the temporal order. The order in which the different elements appear within this operator reflects the order that holds amongst them.
- The ***Relation (R)*** operator should be applied when a relation exists between n elements y_i, i=1,…, n and one element *x*. Referring to the task model, a typical situation is when we have a leaf task *t* at the right side of a disabling operator: all the tasks that could be disabled by t (at whatever task tree level) are in relation with *t*. Again, also this operator is not commutative.
- The ***Hierarchy (H)*** operator means that a hierarchy exists amongst the involved interactors. It is the importance level associated with the operands that identifies the prominence degree that the associated interaction objects should have within the user interface. In order to convey this information, various techniques could be used. In graphical user interfaces one example is allotting within the screen a larger area to objects which are hierarchically more 'important'.

Once the static arrangement of the abstract user interface is identified, also its dynamic behaviour has to be specified, by means of the so-called *transition tasks*, which are tasks whose execution makes the abstract user interface pass from the current presentation into another presentation.

In the next section we will describe more in depth how all those components have been specified in the related DTD of TERESA AUI language.

6.2 The DTD of the TERESA AUI Language

This language can be used for specifying how the various Abstract Interaction Objects (AIO) composing the UI are organised, together with the specification of the dialogue of the UI. For semplicity, we consider and comment just the most relevant parts of TERESA AUI DTD.

The root of the document is the *interface* object. An interface is composed of one or more objects of type presentation:

```
<!ELEMENT interface (presentation+)>
```

Each *presentation* has two parts: the first part (*connection*) is related to the dynamic behaviour of the presentation, the second one (*structure*) is related to the static arrangement of the elements (namely, AIOs) composing the presentation itself. For each presentation, we can have zero, one or more objects of type connection, with each connection mainly identifying the presentation element whose activation allows the interface to move to a different presentation. The structure part mainly describes the static arrangement of the different objects within the presentation itself:

```
<!ELEMENT presentation (connection*, structure)>
<!ATTLIST  presentation name ID #REQUIRED>
```

Each connection has two attributes: an *interaction_aio_id*, which defines the interaction object whose performance triggers the next presentation which is identified in turn by the *presentation_name* attribute:

```
<!ELEMENT        connection EMPTY>
<!ATTLIST  connection
             interaction_aio_id IDREF #REQUIRED
             presentation_name IDREF #REQUIRED>
```

Each structure element can be either an elementary abstract interaction object (*aio*) or a composition of them (*aio_composition*) through the various operators defined in the abstract language:

```
<!ELEMENT        structure (aio | aio_composition)>
```

Each *aio_composition* is the composition of one operator defined in the language (*grouping*, *ordering*, *relation*, *hierarchy*) with a number of expressions which can be, in turn, either elementary interaction objects or complex expression of such elementary objects. Note that the *second_expression* tag is only used when the concerned operator is the relation: in this case we have to model a N:1 relation, so the *second_expression* tag is used to identify precisely the element which the other N elements are in relation with.

```
<!ELEMENT   aio_composition   (operator,   first_expression+,
second_expression?)>
<!ELEMENT        operator EMPTY>
<!ATTLIST        operator name   (grouping | ordering | relation
| hierarchy)  #REQUIRED >
<!ELEMENT        first_expression (aio | aio_composition)>
<!ELEMENT        second_expression (aio | aio_composition)>
```

Each aio can be either an interaction object (*interaction_aio*) or an application object (*application_aio*). In any case, it is univocally identified within the presentation by means of the *id* attribute.

```
<!ELEMENT  aio (interaction_aio | application_aio)>
<!ATTLIST  aio id ID #REQUIRED>
```

Each *interaction_aio* defines an abstract interaction object which implies an interaction between the user and the application. It can be of different types depending on the type of task supported, for example: *selection_aio*, if the object supports selection from a set of elements; *edit_aio*, if it supports editing an object, *control_aio* if it allows triggering an event within the user interface. Each *selection_aio* can be of different type, depending on the number of elements that will be selected. If the element allows a single choice, there are a number of options depending on the cardinality (low/medium/high) of the set which the element will be selected from. The same holds in case of multiple choice.

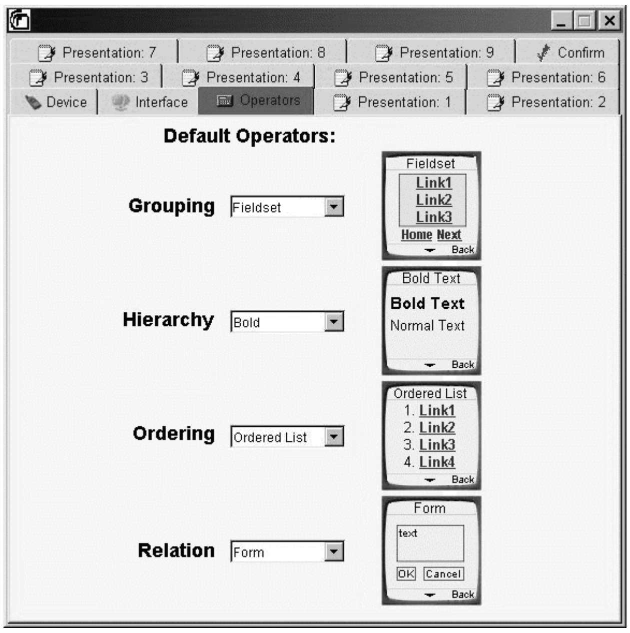
Another type of interaction object is the *edit_aio*, which can be of different types, depending on the type of object that will be edited (such as text, graphic, numerical quantity or position). The last type of interaction object is *control_aio*, which is mainly associated with interaction objects able to trigger a particular 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 associated with different types depending on the type of output the application provides to the user: a textual output, a graphical output, an image, a feedback about a particular state of the user interface.

7. HOW DESIGNERS CAN INTERACT WITH TERESA

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, while in yet others, they want to have full control in order to modify all the possible details in the design process.

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 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 styles, 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 in order to facilitate the understanding of the main characteristics of the resulting design.



Default Operators:

Grouping Fieldset

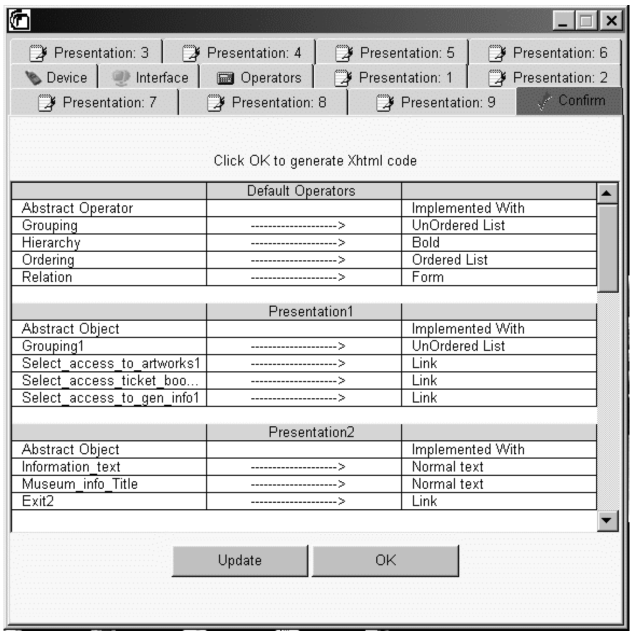
Hierarchy Bold

Ordering Ordered List

Relation Form

Figure 4. Panel for selecting the implementation of each operator of abstract language.

Figure 4 shows how the designer can see the result of the prototyping process. Some control panels are provided to designers in order to change some parameters and an overall summary table is provided by the tool in order to allow designers to understand the design criteria currently applied (an example is shown in Figure 5).



Click OK to generate Xhtml code

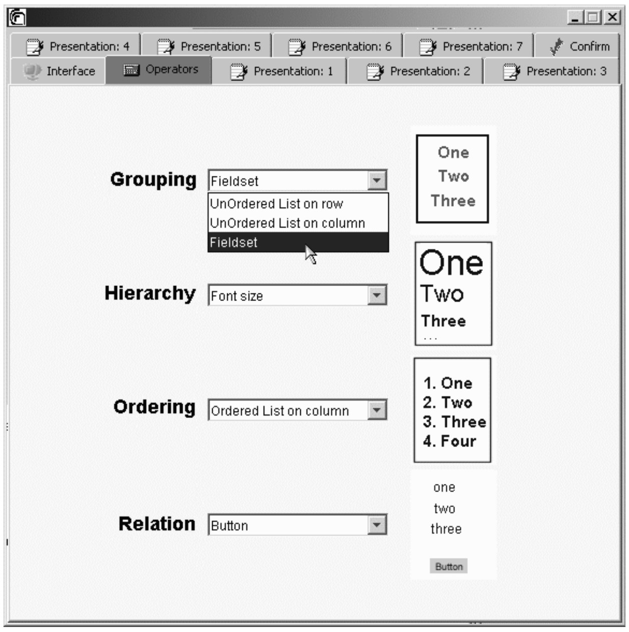
	Default Operators	
Abstract Operator		Implemented With
Grouping	------------------->	UnOrdered List
Hierarchy	------------------->	Bold
Ordering	------------------->	Ordered List
Relation	------------------->	Form

	Presentation1	
Abstract Object		Implemented With
Grouping1	------------------->	UnOrdered List
Select_access_to_artworks1	------------------->	Link
Select_access_ticket_boo...	------------------->	Link
Select_access_to_gen_info1	------------------->	Link

	Presentation2	
Abstract Object		Implemented With
Information_text	------------------->	Normal text
Museum_info_Title	------------------->	Normal text
Exit2	------------------->	Link

Figure 5. Moving from abstract user interface to concrete user interfaces for mobile phones.

The different platforms currently considered allowed us to identify a number of differences between the design for desktop and for mobile applications. For example, as you can see from Figure 6(top)-(bottom), the same grouping operator has been implemented with different techniques depending on whether the desktop or the mobile platform is considered.

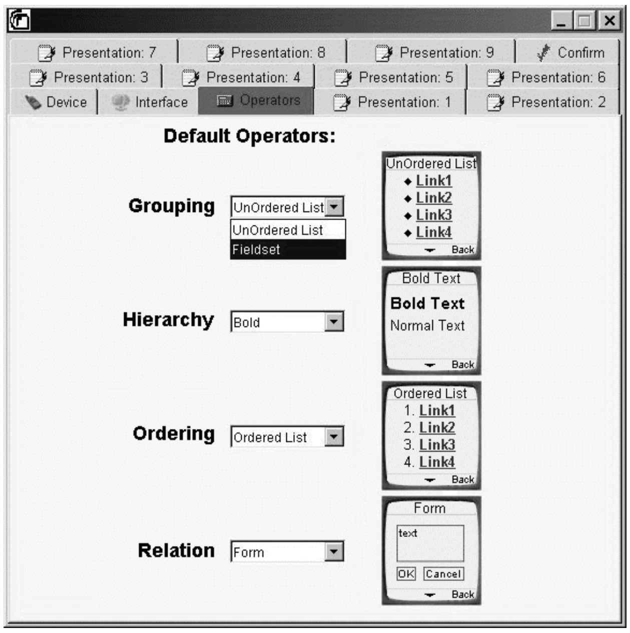


Grouping Fieldset

Hierarchy Font size

Ordering Ordered List on column

Relation Button



Default Operators:

Grouping UnOrdered List

Hierarchy Bold

Ordering Ordered List

Relation Form

Figure 6. (top) Grouping techniques for desktop platform; (bottom) Grouping techniques for mobile platform.

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). 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 (see figure 6-bottom), 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 aio is considered. Neither of them can be used in the mobile environment respectively because in the first case the small area of cellphones does not allow to consider this dimension and, in the second case, the limited capability of the device 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 between the implementation of the various operators between the two platforms, for example in the global parameters that are made 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 device.

8. AN EXAMPLE OF APPLICATION

In this section we will show a sample example of application of the described approach, by considering a museum application. In this domain, the user is supposed to be able to perform a number of activities concerning accessing museum and artwork information, ticket booking, etc. For this example two main platforms have been taken into consideration, a cellphone and a desktop platform, and the related task model has been specified.

Once that the integrated task model has been specified, the first step is to apply the filtering so as to obtain the related task model for each platform considered. For example, if we focus on the *Showing_artwork_info* task, the resulting task models are shown in Figure 7(a)-(b) below.

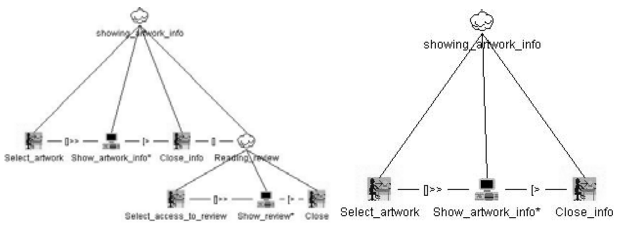


Figure 7. The task *Showing_artwork_info* after having applied the filter for the desktop environment (a) and for the mobile one (b).

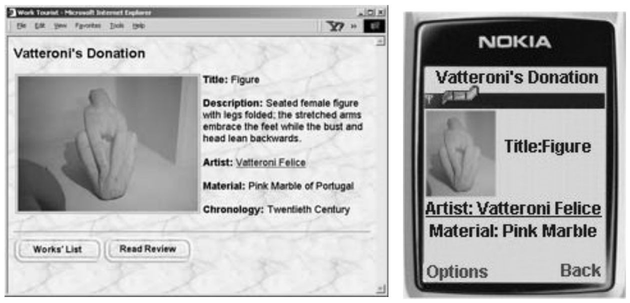
As you can see, at the task level, in the mobile platform the task allowing access to a review (*Reading_review*) has been pruned from the tree. At the object specification level, an example of differences among the supporting platforms has already been shown in Figure 1.

Once the task models of both cellphone and desktop platforms are obtained, the process evolves into two parallel, separate tracks, one for each platform.

For example, the expression of the abstract user interface in the desktop environment is R(H(show_artwork_info, show_section), G(Access_to_list_of_works, Read_review), which means that at the highest level there is a relation operator which puts in relation a hierarchal composition of two elements with a grouping of two other objects. It is worth noting that in the previous example, for simplicity we put the task name instead of the corresponding aio name. However, whenever a task manipulates a number of objects, a grouping composition involving the manipulated objects is generated in the abstract user interface.

Once this expression for the abstract user interface is generated for the desktop environment, the tool provides designers with the possibility of changing some parameters for the user interface generation process. Apart from the possibility of setting some global parameters, the designer can select a specific technique to implement a specific abstract interaction object, if that suggested by the system is not considered the most suitable one.

In Figure 8-a you can see the resulting user interface in the case of desktop device, whereas in the part (b) of the figure the correspondent user interface in the case of a mobile device is shown. We note that in the desktop environment not only a larger number of domain objects are shown, but also additional tasks are available (for example the possibility of accessing a review of the artwork).


Vatteroni's Donation

Title: Figure

Description: Seated female figure with legs folded; the stretched arms embrace the feet while the bust and head lean backwards.

Artist: Vatteroni Felice

Material: Pink Marble of Portugal

Chronology: Twentieth Century

Vatteroni's Donation

Title:Figure

Artist: Vatteroni Felice

Material: Pink Marble

Work Tourist - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Works' List
Read Review
NOKIA
Options
Back

Figure 8. Resulting user interface of the example in the desktop environment (a) and in the mobile one (b).

9. CONCLUSIONS

We have presented a tool supporting design and development of nomadic applications. It allows designers to provide the results of conceptual analysis in terms of tasks and their relations and support user interface generation while taking into account the characteristics of the platform considered. Designers have different levels of control over the development process.

The tool is publicly available at http://giove.cnuce.cnr.it/teresa.html

While the current TERESA version supports the design and development of graphical interfaces for various platforms, further work will be dedicated to support multimedia interfaces for a broader set of mobile devices including vocal interaction techniques.

10. ACKNOWLEDGMENTS

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

11. REFERENCES

[1] Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J. *UIML: An Appliance-Independent XML User Interface Language*, Proceedings of the 8th WWW conference, 1999. Available at http://www.harmonia.com/resources/papers/www8_0599/index.htm

[2] 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.

[3] 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.

[4] 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, August 2002 (Vol. 28, No. 8).

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

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

[7] 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, May 2002.

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

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

[10] 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.

[11] Szekely, P., Sukaviria, P., Castells, O., Muthukumarasamy, J., Salcher, E., *Declarative Interface Models for UserInterface 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.

[12] Thevenin D., *ARTStudio; Tool for Multi-target UI Design*, Poster at UIST 2002, Paris.