The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels

Silvia Berti, Francesco Correani, Fabio Paternò, Carmen Santoro

{silvia.berti, francesco.correani, fabio.paterno, carmen.santoro}@isti.cnr.it
ISTI-CNR
Via G.Moruzzi 1
Pisa, Italy

ABSTRACT

The purpose of this paper is to report on the use of XML languages to support the TERESA tool. This is a tool for model-based design of multi-device interfaces. It considers three levels of abstractions (task model, abstract user interface and concrete user interface). For each of them a specific language has been defined and used. In addition, since the lowest abstract level (the concrete interface) is platform-dependent, there are different variants for each platform considered.

Keywords
Model-based design, XML user interface languages, tools.

INTRODUCTION

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

Model-based approaches [4] could represent a feasible solution for addressing such issues: the basic idea is to identify useful abstractions highlighting the main aspects that should be considered when designing effective interactive applications. Our approach extends previous work in the model-based design area in order to support development of nomadic applications. In particular, 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 interfaces for different types of devices from logical descriptions. It differs from other approaches such as UIML [1], which mainly consider low-level models. XIML [6] has similar goals but there is no publicly available tool supporting it.

THE METHOD

Our method for model-based design is composed of a number of steps that allow designers to start with an overall envisioned task model of a nomadic application and then derive concrete and effective user interfaces for multiple devices:

- *High-level task modelling of a multi-context application*. In this phase designers develop a single model that addresses the possible contexts of use and the various roles involved and also a domain model aiming to identify all the objects that have to be manipulated to perform tasks and the relationships among such objects. Such models are specified using the ConcurTaskTrees (CTT) notation [4], which also allows designers to indicate the platforms suitable to support each task.
- *Developing the system task model for the different platforms considered*. Here designers have to filter the task model according to the target platform and, if necessary, further refine the task model, depending on the specific device considered, thus, obtaining the system task model for the platform considered.
- *From system task model to abstract user interface*. Here the goal is to obtain an abstract description of the user interface composed of a set of presentations that are identified through an analysis of the task relationships. Each presentation is structured by means of interactors composed of various operators.
- *User interface generation*. In this phase we have the generation of the user interface. This phase is completely platform-dependent and has to consider the specific properties of the target device.

THE TOOL

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

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

- *Mixed initiative*; we want a tool able to support different levels of automation ranging from completely automatic solutions to highly interactive solutions where designers can tailor or even radically change the solutions proposed by the tool.
- *Model-based*, the variety of platforms increasingly available can be better handled through some abstractions that allow designers to have a logical view of the activities to support.
- *XML-based*, each abstraction level considered can be described through an XML-based language.
- *Top-down*, this approach is an example of forward engineering. So, designers first have to create more logical descriptions, and then move on to more concrete representations until the final interface is obtained.
- *Different entry-points*, our approach aims to be comprehensive and to support various possibilities, including also when different set of tasks can be performed on different platforms. However, there can be cases where only a part of it needs to be supported and, for example, designers want to start with a logical interface description and not with a task model.
- *Web-oriented*, we decided that Web applications should be our first target. However, the approach can be easily extended to other environments (such as Java applications, Microsoft environments, …) by just modifying only the last transformation (from concrete interface to final interface).

The TERESA tool offers a number of transformations and provide designers with an integrated environment for generating XHTML interfaces for desktop, mobile phones and VoiceXML user interfaces. With the TERESA tool, at each abstraction level the designer is in the position of modifying the representations while the tool keeps maintaining forward and backward the relationships with the other levels. For example, it maintains links between abstract interaction objects and the corresponding tasks in the task model so that designers can immediately identify their relations. This results in a great advantage for designers in maintaining a unique overall picture of the system, with an increased consistence among the user interfaces generated for the different devices and consequent improved usability for end-users.

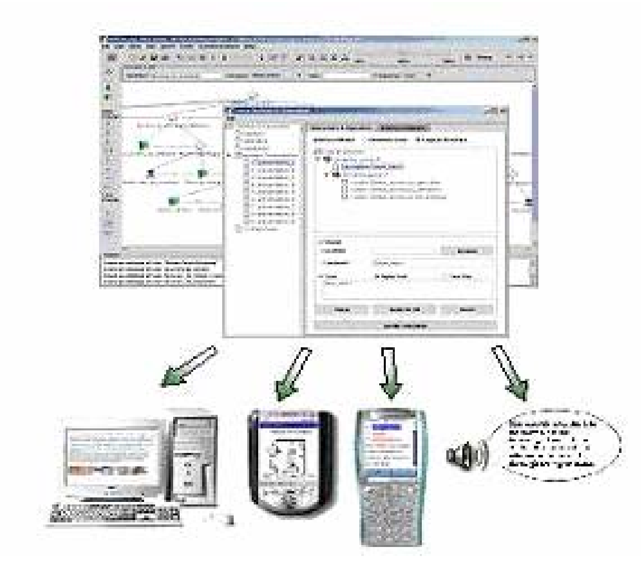


Figure 1: The TERESA tool.

Once the elements of the abstract user interface have been identified, every interactor has to be mapped into interaction techniques supported by the particular device configuration considered (characterised by the modalities supported, the screen size, …), and also the abstract operators have to be appropriately implemented by highlighting their logical meaning: a typical example is the set of techniques for conveying grouping relationships in visual interfaces by using presentation patterns like proximity, similarity and continuity. However, different techniques for grouping elements are used in case of vocal interfaces, such as using a specific sound to delimit a set of elements.

How XML-based Languages are Used in the Tool

TERESA is a transformation-based tool that supports the design of an interactive application at different abstraction levels and generates the concrete user interface for various types of platforms. By platform we mean a class of systems that share the same characteristics in terms of interaction resources. Such transformations exploit a number of XML languages. The main transformations supported in TERESA are:

- *Presentation task sets and transitions generation*. From the XML specification of a CTT task model concerning a specific platform, it is possible to obtain the Presentation Task Sets (PTSs), sets of tasks which are enabled over the same period of time according to the constraints indicated in the model and transitions specifying the conditions allowing moving across PTSs. Such sets, depending on the designer's application of a number of heuristics (general criteria used to merge together two or more PTSs) supported by the tool, can be grouped together so identifying the groups of tasks that should be supported by each user interface presentation.

- *From task model -related information to abstract user interface*. The goal of this phase is mapping the task-based specification of the system onto an interactor-based description of the related abstract user interface. Both the XML task model and Presentation Task Sets specifications are the input for the transformation generating the associated abstract user interface. Currently, each basic task that manipulates a user interface element is associated with an interactor in the abstract 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 is particularly useful especially when it comes to specifying the properties of each interactor, as the knowledge of the task it supports is an important indication of its meaning and goal, so it helps designers to position the interactor within the overall application and decide on the most appropriate settings.
- *From abstract user interface to concrete interface for the specific platform*. This transformation starts with the loading of a XML abstract user interface specification previously saved and yields the related concrete user interface for the specific media and interaction platform selected, which is saved in the associated XML language. Currently, there is a one-to-one mapping between abstract and concrete interactors. A number of parameters related to the customisation of the concrete user interface are made available to the designer.
- *Automatic UI Generation*. The tool automatically generates the final UI for the target platform. The starting point can be the single-platform task model, using a number of default configuration settings related to the user interface generation, or the abstract or the concrete user interface.

THE TASK MODEL

The task model is represented by the ConcurTaskTrees (CTT) notation [4], which 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), as well as a number of additional attributes (such as frequency) (see Figure 3).

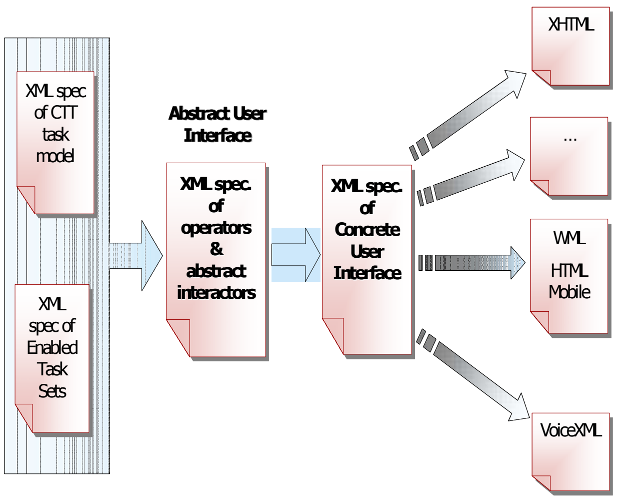

XML spec of CTT task model
XML spec of Enabled Task Sets
Abstract User Interface
XML spec. of operators & abstract interactors
XML spec. of Concrete User Interface
XHTML
...
WML
HTML Mobile
VoiceXML

Figure 2: How XML Languages are used in TERESA.

The notation is tool supported. Initially the idea was to facilitate the development of task models and support saving them in XML format. It was the first notation for task models described in XML format (this feature has been available since 1998). It is currently used also in other environments (such as [2]) for user interface design and development.

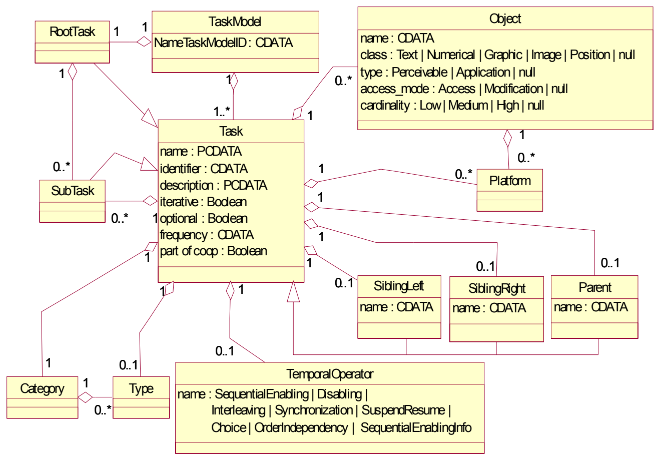

RootTask
TaskModel
NameTaskModelID : CDATA
Object
name : CDATA
class : Text | Numerical | Graphic | Image | Position | null
type : Perceivable | Application | null
access_mode : Access | Modification | null
cardinality : Low | Medium | High | null
Task
name : PCDATA
identifier : CDATA
description : PCDATA
iterative : Boolean
optional : Boolean
frequency : CDATA
part of coop : Boolean
SubTask
Platform
SiblingLeft
name : CDATA
SiblingRight
name : CDATA
Parent
name : CDATA
Category
Type
TemporalOperator
name : SequentialEnabling | Disabling | Interleaving | Synchronization | SuspendResume | Choice | OrderIndependency | SequentialEnablingInfo
1
1..*
0..*
0..1

Figure 3: Class diagram representing the concepts of the ConcurTaskTrees notation.

In order to support this approach we needed to extend the notation 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. It is worth noting that at this level –the task level- sets of devices sharing certain similarities are considered, rather than *specific* devices. So, in our framework we provide for typical sample device clusters as mobile phones and PDAs are, together with the possibility for designers to define their own platforms. This has proved to be both feasible and flexible to tackle the problem of dealing with the disparate devices that our approach has to consider. Nevertheless, additional levels of refinement within the same cluster are considered in the last phase of the method, when knowing the specific characteristics of the devices considered becomes useful for producing effective final user interfaces.

The platform attribute has also been associated with the objects manipulated during task accomplishment. Indeed, CTT allows designers to specify for each task what objects should be manipulated during its performance.

THE ABSTRACT USER INTERFACE

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

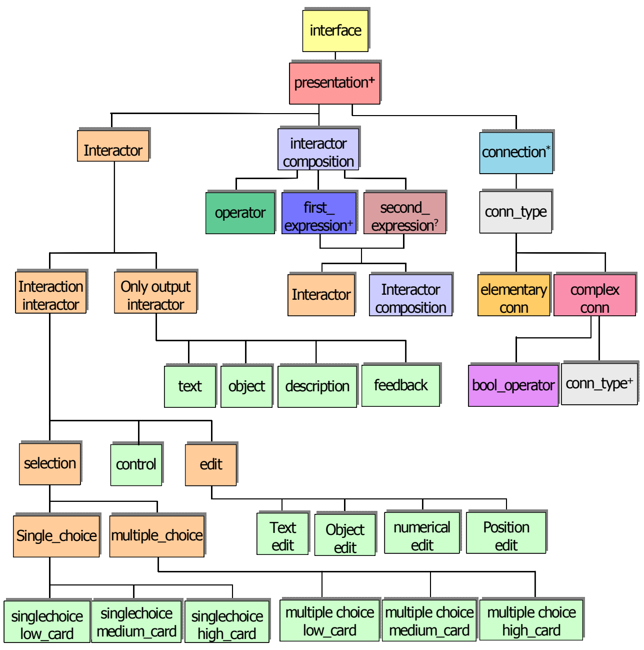


Figure 4: The Structure of the Abstract User Interface Language.

The structure of the presentation is defined in terms of interactors (abstract descriptions of interaction objects classified depending on their semantics) [5] and their composition operators (see Figure 4). It is possible to distinguish between interactors supporting user interaction (interaction elements) and those that present results of application processing (only_output elements). The interaction elements imply an interaction between the user and the application. There are different types of interaction elements depending on the type of task supported. We have selection elements (to select between a set of elements), edit (to edit an object), control (to trigger an event within the user interface, which can be useful to activate either a functionality or the transition to a new presentation). Differently, an only_output element defines an interactor which implies an action only from the application. There are different types of only_output elements (text, object, description, feedback) depending on the type of output the application provides to the user: a textual one, an object, a description, or a feedback about a particular state of the user interface.

The composition operators can involve one or two expressions, each of them can be composed of one, 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 [3]. 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.

CONCRETE GRAPHICAL INTERFACE

In this section we describe the XML language for two graphical platforms: the desktop and the mobile phone. In both cases the structure is designed to be similar to the structure of the abstract user interface language. In this way, passing through these levels in the UI generation process, we are always able to easily recognize the interactor hierarchy.

Implementation details are mostly provided by a deeper tree representation, with leaf nodes defining concrete information. Differences among concrete user interface specifications belonging to different devices, thus, can be mainly found in the lowest levels of the hierarchical structure.

A concrete desktop user interface is defined by a number of presentations and default settings to be used in the generation phase.

```
<!ELEMENT concrete_desktop_interface
(default_settings, presentation+) >
```

```
<!ELEMENT default_settings (background, font_settings,
operators_settings, interactors_settings)>
```

Mobile devices also require specifying further data regarding expressive capabilities of the target phone.

```
<!ELEMENT concrete_mobile_interface (device_type,
default_settings, presentation+)>
<!ELEMENT device_type (big | medium | small)>
<!ELEMENT big EMPTY>
<!ATTLIST big graphic_support (%option;)
#REQUIRED>
<!ELEMENT medium EMPTY>
<!ATTLIST medium graphic_support (%option;)
#REQUIRED>
<!ELEMENT small EMPTY>
<!ATTLIST small graphic_support CDATA #FIXED
"no">
```

As usual, each presentation contains information about interactor organization and connections to other presentations, but specific properties, like title, header etc., are also described.

```
<!ELEMENT presentation (presentation_properties,
connection*, (interactor | interactor_composition))>
<!ELEMENT presentation_properties (title, background,
font_settings, top)>
```

Connection and interactor composition are defined as in the AUI language for each type of target platform.

```
<!ELEMENT connection (conn_type)>
<!ATTLIST connection presentation_name IDREF
#REQUIRED>
<!ELEMENT conn_type (elementary_conn |
complex_conn)>
[…]
<!ELEMENT interactor (interaction | only_output)>
<!ELEMENT interactor_composition (operator,
first_expression+, second_expression?)>
<!ELEMENT operator (grouping | ordering | hierarchy |
relation)>
```

In the Interactor specification the deeper we proceed, the more concrete details we get. For instance, let us consider how we can describe single selection interactors with our desktop CUI notation.

```
<!ELEMENT interaction (selection | editing | control)>
<!ELEMENT selection (single | multiple)>
<!ELEMENT single (radio_button | list_box |
drop_down_list)>
<!ATTLIST single cardinality (%cardinality_value;)
#REQUIRED>
<!ELEMENT radio_button (choice_element+)>
<!ATTLIST radio_button label CDATA #REQUIRED>
<!ELEMENT choice_element EMPTY>
<!ATTLIST choice_element
        label CDATA #REQUIRED
        value CDATA #REQUIRED>
[…]
```

Differences among other CUIs consist of different concrete elements associated to a given type of interactor; e.g. let us consider the previous example for mobile devices, list boxes are not usable in this context so they are not allowed.

```
<!ELEMENT single (radio_button | drop_down_list)>
[…]
```

Each CUI formalizes the expressive power of a given device type in terms of concrete interactors and operators available in that platform. While in desktop environments grouping operators can be implemented by combining several techniques, in mobile phones, because of the limited screen dimensions, we can choose only one implementation technique from a limited set.

It can happen that some abstract interactor is related to the same concrete elements even in different CUI. In this case, differentiation is granted by allowing different attributes values for the same concrete object.

For instance we can refer to the text edit objects: they can be implemented with a text field in both desktop and mobile cases.

```
<!ELEMENT text_edit (textfield)>
<!ELEMENT textfield EMPTY>
<!ATTLIST textfield
        label CDATA #REQUIRED
        length (%length_value;) #REQUIRED
        password (%option;) #REQUIRED>
```

However, the reduced screen capabilities of mobile devices are considered, thus allowing lower length values.

Desktop CUI

<!ENTITY % length_value "8 | 9 | 10 | 11 | 12 |13 | 14 | 15 | 16 | 17 | 18 |19 | 20">

Mobile CUI

<!ENTITY % length_value "4 | 5 | 6 | 7 | 8 | 9 | 10">

CONCRETE VOCAL INTERFACE

Like graphical interfaces, in vocal interfaces a concrete user interface is composed of some default settings and a set of presentations corresponding to the presentations of the abstract user interface language. The difference is that in this language, interactors and their compositions are obtained through techniques specific for the vocal interface, so they have different attributes.

The default settings are applied to the entire vocal application and are important for supporting the user interactions.

```
<!ELEMENT default_settings (name_application, welcome_msg, def_commands?, synthesis_properties, recognition_properties, bargein, operator_settings+)>
<!ELEMENT name_application EMPTY>
<!ATTLIST name_application value CDATA #REQUIRED>
```

The welcome message allows users to understand the current context and that they are talking to a computer that accepts a well defined language. In this case is possible to use some default message (short, medium or long) or to define a new message (new). Another useful parameter of this element is onlyOnce that allows skipping a welcome message when the user visits the main presentation for the second time.

```
<!ELEMENT welcome_msg EMPTY>
<!ATTLIST welcome_msg
        type (short | normal | long | new) #REQUIRED
        msg CDATA #REQUIRED
        onlyOnce (%boolean;) #REQUIRED>
```

Other general settings regard: the property of synthesis or recognition engine, the barge-in option that allows the user to interrupt a prompt in order to speed up the dialog sequence and some default commands that allow users to disable the device input and/or to exit from voice application.

```
<!ELEMENT synthesis_properties EMPTY>
<!ATTLIST synthesis_properties
        pitch (%pitch_value;) #REQUIRED
        rate (%rate_value;) #REQUIRED
        volume(%volume_value;) #REQUIRED>
<!ELEMENT recognition_properties EMPTY>
<!ATTLIST recognition_properties
        confidence CDATA #REQUIRED
        sensitivity CDATA #REQUIRED
        completetimeout CDATA #REQUIRED
        incompletetimeout CDATA #REQUIRED>

<!ELEMENT def_commands (exit?, disable?)>
<!ELEMENT disable EMPTY>
<!ATTLIST disable
        cmd_dis CDATA #REQUIRED
        cmd_activ CDATA #REQUIRED>
<!ELEMENT exit EMPTY>
<!ATTLIST exit  msg_to_exit CDATA #REQUIRED>
<!ELEMENT bargein EMPTY>
<!ATTLIST bargein active (%boolean;) #REQUIRED>
```

The structure of each presentation is defined in terms of some general properties similar to those considered in the default settings but in this case they concern one specific presentation, the dynamic behaviour of the user interface, the vocal properties of interactors, and the vocal techniques exploited to represent the composition operators.

```
<!ELEMENT presentation (presentation_properties, connections*, (interactor | interactor_composition))>
<!ATTLIST presentation name ID #REQUIRED>
```

For example, the vocal properties of selection interactor concerns three types of menu:

```
<!ELEMENT selection (single | multiple)>
<!ELEMENT single (dtmf_menu | enumerate_menu | message_menu)>
<!ELEMENT multiple (dtmf_menu | enumerate_menu | message_menu)>
```

The meaning of the possible values is:

- <dtmf_menu>: in this case in the vocal interface the user can perform the selection only through the keypad and so the message of synthesizer will be "If you want a coffee, press 1; if…"
- <enumerate_menu>: in this case in the vocal interface the synthesizer produce a list of option as "Do you want: coffee, tea, milk…".
- <message_menu>: in this case in the vocal interface the synthesizer generate an elaborated message like "if you prefer coffee, say coffee; if…"

In any case, the selection interactors can define a feedback message to confirm if a command is correctly understood or how to manage some events such as no input or no match or help or define a reply message in order to listen again the choices.

In the concrete user interface composition operators can be represented through different vocal techniques according to their logical meaning and communication goals. Grouping can be represented through four techniques:

```
<!ELEMENT grouping (Insert_sound | Insert_pause | Change_volume | Keywords)>
<!ELEMENT Insert_sound EMPTY>
<!ATTLIST Insert_sound
        src_audio_file CDATA #REQUIRED>
```

The technique inserts a sound at the beginning and at the end of the grouped elements and, in this case, the audio file is specified.

```
<!ELEMENT Insert_pause EMPTY>
<!ATTLIST Insert_pause
        lenght_pause CDATA #REQUIRED>
```

This technique inserts a pause at the end of the grouped elements and in this case is specified the duration of pause.

```
<!ELEMENT Change_volume EMPTY>
```

A specific volume can be used during the speech synthesis of the grouped elements.

```
<!ELEMENT Keywords EMPTY>
```

The keywords technique inserts some words to highlight the grouping operator (for examples: "In this Application you can choose one of this option: If you would like some general information, say information… if you would like to book a ticket, say ticket. Alternatively if you would…").

Ordering can be represented by two techniques: arranging objects in alphabetical order, and keywords techniques that insert some words to highlight the operator order (for example: "In this presentation at the beginning you should say a name, after the sound say password and lastly say go in order to proceed").

```
<!ELEMENT ordering (Arrange_objects_in_alphabetical_order | Keywords)>
<!ELEMENT Arrange_objects_in_alphabetical_order EMPTY>
<!ELEMENT Keywords EMPTY>
```

The Relation operator supports a vocal input that enables a change in context by moving to another presentation. This type of operation can be useful to navigate within the presentation.

```
<!ELEMENT relation (Change_context)>
<!ELEMENT Change_context EMPTY>
```

The Hierarchy operator is represented through two techniques: increasing or decreasing the volume of the synthesized voice.

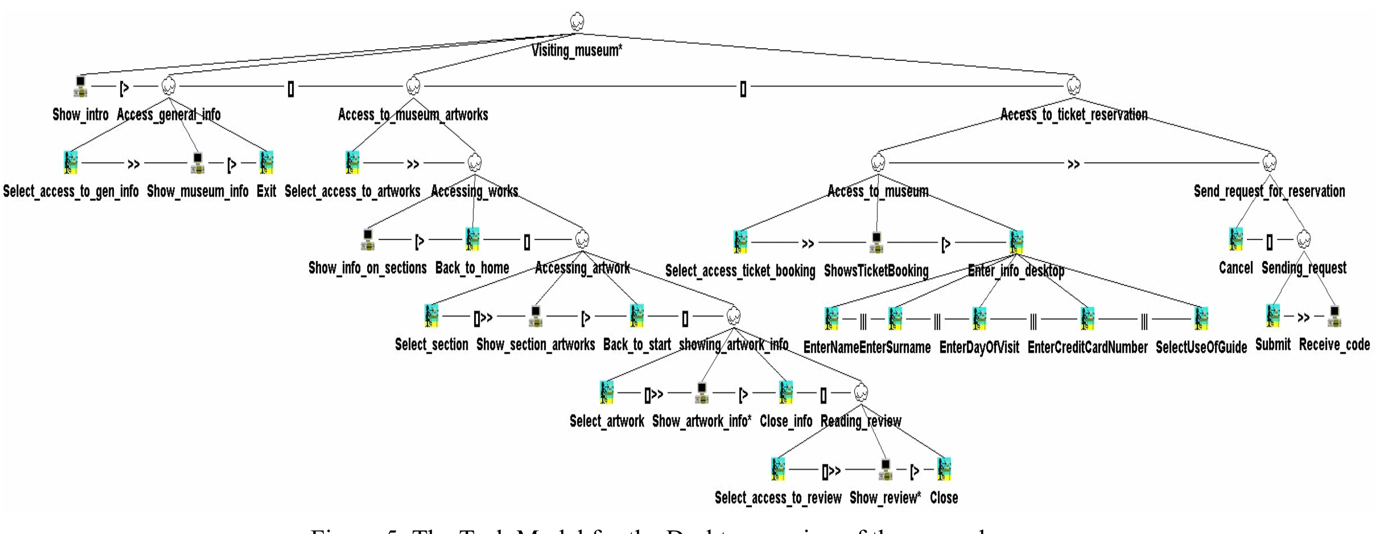


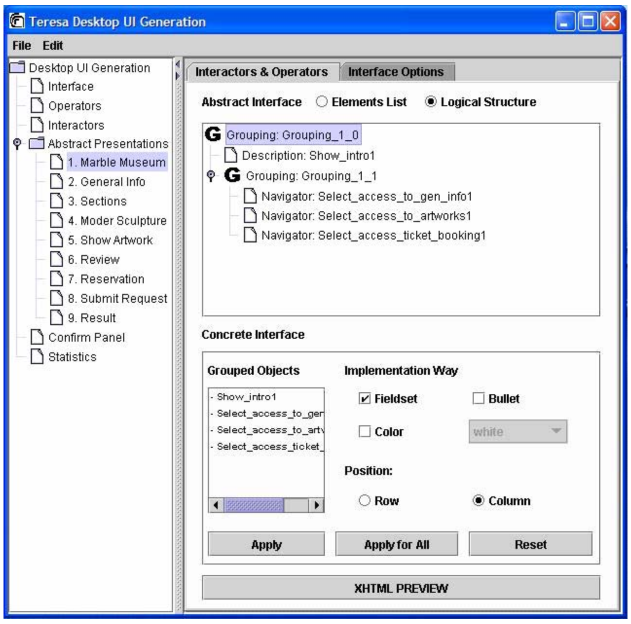
Figure 5: The Task Model for the Desktop version of the example.

AN EXAMPLE

In this section we show an example of this approach in order to better understand how the various abstraction levels are exploited. We consider a museum application.

Figure 5 shows the task model for the desktop version. After the automatic transformation of the task model first into an abstract user interface and then into a concrete user interface, some attributes have been edited in order to obtain the concrete version for a desktop system and we obtain the result shown in Figure 6.

It is possible to see that the user interface is structured into nine presentations automatically identified. The structure of the first presentation is currently presented by the tool. There are two grouping compositions of interactors: one aims to group three navigator interactors allowing the access to the various parts of the application, the second one groups the first grouping with a description interactor. The element currently selected in the control panel is the second grouping interactor. The associated attributes and the corresponding values are displayed in the bottom part.



```
Desktop UI Generation
  Interface
  Operators
  Interactors
  Abstract Presentations
    1. Marble Museum
    2. General Info
    3. Sections
    4. Moder Sculpture
    5. Show Artwork
    6. Review
    7. Reservation
    8. Submit Request
    9. Result
  Confirm Panel
  Statistics
```

Abstract Interface ○ Elements List ● Logical Structure

```
G Grouping: Grouping_1_0
  Description: Show_intro1
  G Grouping: Grouping_1_1
    Navigator: Select_access_to_gen_info1
    Navigator: Select_access_to_artworks1
    Navigator: Select_access_ticket_booking1
```

Grouped Objects: Show_intro1; Select_access_to_ger; Select_access_to_artw; Select_access_ticket_

Implementation Way: ☑ Fieldset ☐ Bullet ☐ Color white

Position: ○ Row ● Column

Figure 6: The Structure of the User Interface of the example.

Figure 7 shows the corresponding user interface. In the bottom part we can see the navigator interactors which are lined up in a horizontal manner with consistent appearance, thus implementing the grouping operator. In the top part there is the implementation of the description element aiming to introduce the Marble Museum and its artworks. The resulting interface implements the indications contained in the task model. Indeed, in the model the first is a system task aiming to introduce the museum showing the home page. Such task can be disabled by three interactive tasks whose purpose is to enable different types of high-level tasks (access to general information, artworks, and ticket reservation). These four basic tasks are those associated with the first presentation in the abstract and concrete user interface.

CONCLUSIONS

The TERESA environment supports design and development of multi-platform user interfaces through a number of transformations that can be performed either automatically or through interactions with the designer.

To this end, a number of XML languages that capture the relevant information at different abstraction levels are used. Such languages are introduced in this paper along with a discussion of how they are used in the environment.

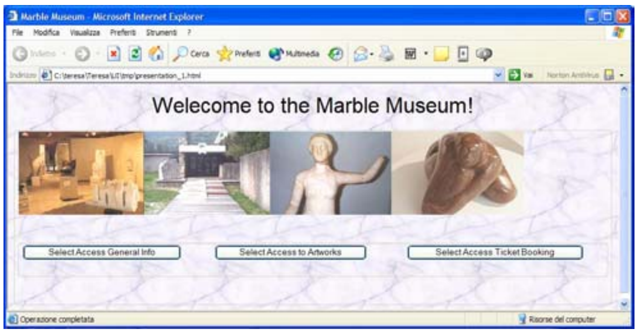

Marble Museum - Microsoft Internet Explorer
Welcome to the Marble Museum!
Select Access General Info
Select Access to Artworks
Select Access Ticket Booking

Figure 7: The interface of the first page of the example.

The tool can be freely downloaded at http://giove.cnuce.cnr.it/teresa.html.

Future work will be dedicated to supporting generation in further multimodal user interface languages. We also plan to support importing of CAMELEON XML representations in order to ease exchange of information with other tools and facilitate integration of forward and reverse engineering environments

ACKNOWLEDGMENTS

We gratefully acknowledge support from the EU IST CAMELEON project (http://giove.cnuce.cnr.it/cameleon.html) and the EU SIMILAR NoE (www.similar.cc).

REFERENCES

1. Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J. *UIML: An Appliance-Independent XML User Interface Language*, Proceedings of the 8th WWW conference, 1999.

2. K.Luyten, K.Conix, An XML-based runtime user interface description language for mobile computing devices. Proceedings DSV-IS 2001, pp.20-29, Springer Verlag.

3. Mullet, K., Sano, D., Designing Visual Interfaces. Prentice Hall, 1995.

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

5. 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.

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