A UNIFIED METHOD FOR DESIGNING INTERACTIVE SYSTEMS ADAPTABLE TO MOBILE AND STATIONARY PLATFORMS

Fabio Paternò, Carmen Santoro
CNR
Via G. Moruzzi 1 56100 Pisa Italy
{f.paterno, c.santoro}@cnuce.cnr.it

Abstract The wide variety of devices currently available, which is bound to increase in the coming years, poses a number of issues for the design cycle of interactive software applications. Model-based approaches can provide useful support in addressing this new challenge. In this paper we present and discuss a method for the design of nomadic applications showing how the use of models can support their design. The aim is to enable each interaction device to support the appropriate tasks users expect to perform and designers to develop the various device-specific application modules in a consistent manner.

Keywords: Model-based design, Multi-platform interactive systems, Context of use.

1. INTRODUCTION

Recent years have seen the introduction of many types of computers and devices (e.g. cellphones, PDA's, WebTV, etc.) and the availability of such a wide range of devices has become a fundamental challenge for designers of interactive software systems (Olsen, 1998). Users wish to be able to seamlessly access information and services regardless of the device they are using, even when the system or the environment changes dynamically. To this end, computer-based applications need to run on a wide spectrum of devices.

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 dynamically respond to changes in context. Creating different versions of applications for different devices engenders extra development and expensive maintenance cost of cross-platform consistency, complicates the problems of configuration management and dilutes the resources available for usability engineering. Additionally, current development tools provide little support for creating applications that change dynamically in response to changes in their environment, or that have to share data amongst heterogeneous device types. Although many dimensions must be considered when designing context-dependent applications (actors, platforms, environments, system resources, etc.), in this paper we focus on how to support a design addressing change of platforms.

Nowadays companies need to better consider the thorny issues involved in designing applications supporting multiple platforms. Currently, they often address different contexts of use by building different applications, even if supporting similar tasks accessing similar data. Different versions of applications should differ in their structure, rather than trying to address different input/output platforms by just resizing the elements of the user interface. Versions may also differ in the tasks they can realistically support –for example, the PDA permits reading reviews, the phone permits ordering books, and the desktop PC supports both. In addition the set of functionality supported can change dynamically.

Technological advances solve some of the problems of engineering applications for multiple devices: XML (eXtensible Markup Language) documents supported by XSL (eXtensible Stylesheet Language) stylesheets allow creating customised presentations for different devices or users. Wireless Markup Language (WML) permits to produce device-independent presentations for a range of small display devices. Wireless Internet gateways automatically translate HTML documents into WML documents (although they may produce unusable results if they rely on large displays). However, XSL (and related technologies) help with user interface presentation, but are limited by the fact that a different interaction design may be necessary when moving between radically different device types. More generally, while these solutions help with parts of the problem, they do not provide high-level guidance for guaranteeing quality across multiple versions of applications.

The goal of this paper is to present and discuss a method to support design and development of highly usable context-sensitive interactive software systems. To this end we discuss the state of art in addressing issues related to multi-platform applications. Next, we identify the main phases of the proposed method and also illustrate how the information provided by task models can be useful for the design of user interfaces in multi-platform applications. Then, we analyse more in depth the method and show its application to an example taken from the design of a museum application. Lastly, some concluding remarks are provided.

2. RELATED WORK

In a recent paper, discussing the future of user interface tools, Myers, Hudson, and Pausch (2000) indicate that the wide platform variability *encourages a return to the study of some techniques for device-independent user interface specification, so that developers can describe the input and output needs of their applications, so that vendors can describe the input and output capabilities of their devices, and so that users can specify their preferences. Then, the system might 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 that adapts to its features and possible contexts of use. This is also called user interface plasticity (Thevenin and Coutaz, 1999). Methods for modelling work context (Beyer and Holtzblatt, 1998) can provide useful information for this type of approach.

This problem is a novel challenge for model-based design and development of interactive applications. The potentialities of these approaches have been addressed in a limited manner. In the GUITARE Esprit project a user interface generator was developed: it takes ConcurTaskTrees (CTT) task models (Paternò, 1999) 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 (Puerta, 1997) is an example of a semi-automatic approach but it only supports design of traditional graphical desktop applications.

UIML (Abrams et al., 1999) 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 in 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.

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 (Eisenstein et al., 2001; Calvary et al., 2001), but there are still many issues that should 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.

Various models have been proposed to highlight important aspects in the design of user interfaces (Johnson et al., 1993; Paternò, 1999; Sukavirija et al., 1993; Szekely et al., 1993; Vanderdonckt and Bodart, 1993). 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.

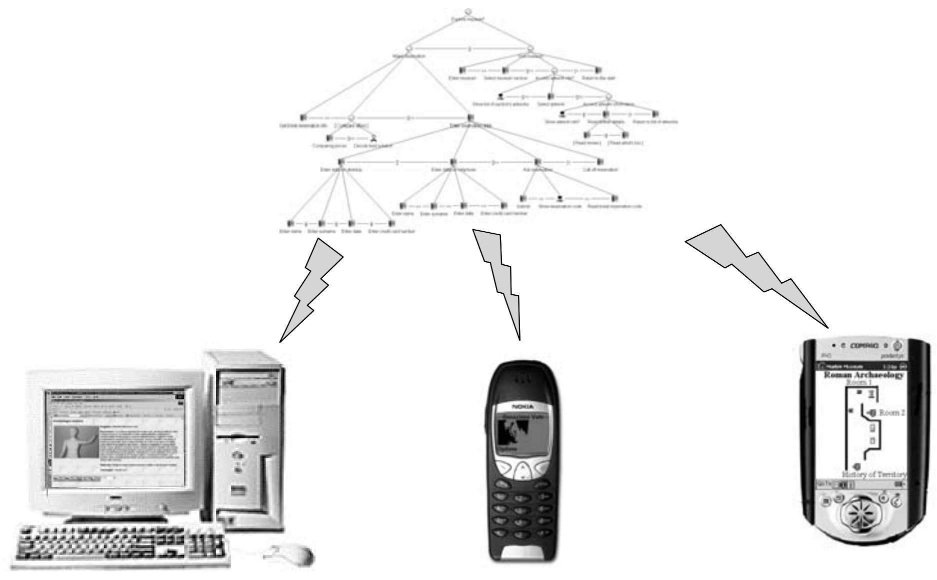


Figure 1: One model, many interfaces.

3. THE PROPOSED METHOD

The design of multi-platform applications can follow different approaches. It is possible to support the same type of tasks with different devices. In this case, what has to be changed is the set of interaction and presentation techniques to support information access while taking into account the resources available in the device considered. However, in some cases designers should consider different devices also with regard to the choice of the tasks to support. For example, phones are more likely to be used for quick access to limited information, whereas desktop systems better support browsing through large amounts of information. To complicate matters, it must be borne in mind that even within the same class of devices there are different presentation models that need to be handled. For example, more and more, cellular phones are being used to access remote applications, and currently access is provided by WAP phones. There are many usability issues that are limiting their spread. While in desktop systems we have mainly two well-known browsers with some compatibility issues (even though such issues often create some problems), in WAP-enabled phones a number of microbrowsers tend to accept slightly different versions of WML, assume to interact with slightly different phones (for examples, phones with a different number of softkeys) and interpret the softkeys interactions differently.

Our method tries to address such problems, and is composed of a number of steps (see Figure 2) that allows 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 need to think about the logical activities that have to be supported and the relationships among them. They develop a single model that addresses the various 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. The CTTE (CTT Environment) tool (publicly available at http://giove.cnuce.cnr.it/ctte.html) supports editing and analysis of task models specified using this notation. The tool allows designers to explicitly indicate the platforms suitable to support performance of 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. This filter-and-refine process in some cases involves creating task

models in which the tasks that cannot be supported in a given platform are removed and the navigational tasks deemed necessary to interact with the considered platform are added. In other cases it forces to add supplementary details on how a task is decomposed when a specific platform is considered. Thus, we obtain 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 abstract presentations that are identified through an analysis of the task relationships and structured by means of interactors composed of various operators. Then, still considering the temporal relationships among tasks, we identify the possible transitions among the user interface presentations considering the temporal relationships that the task model indicates. Analysing task relationships can be useful for structuring the presentation. For example, the hierarchical structure of the task model can be considered to identify interaction techniques to be grouped, for example, those that have the same parent task and are thus logically more related to each other. Likewise, concurrent tasks that exchange information can be better supported by highly integrated interaction techniques (to some extent, merged), as happens when using adjacent techniques, so that users can better follow their mutual dependencies.
- *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. For example, if the considered device is a cellular phone, such information is not sufficient, as we also need to know the type of micro-browser supported and the number and the types of soft-keys available.

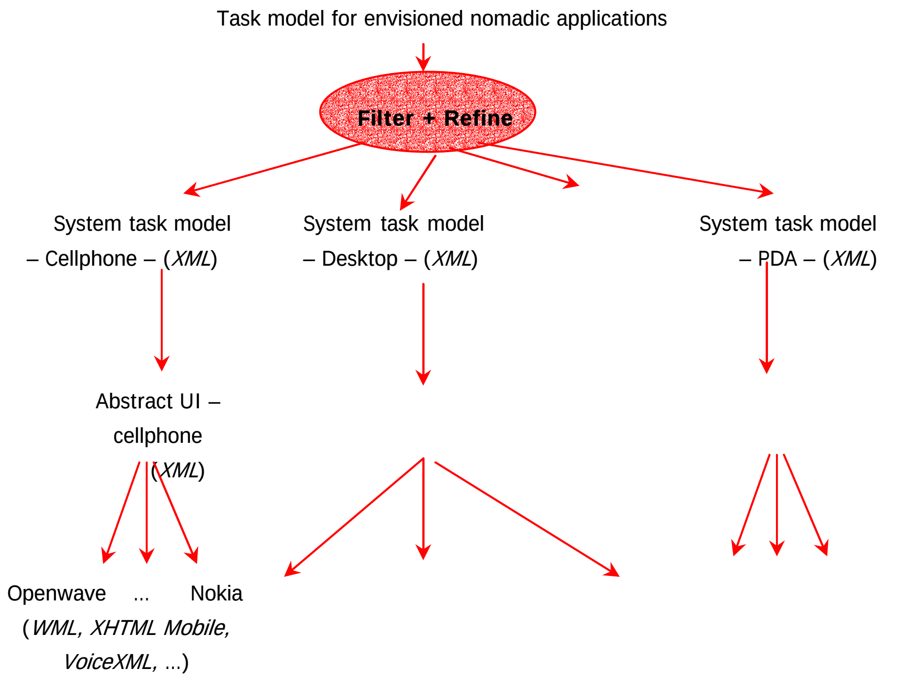

Task model for envisioned nomadic applications
Filter + Refine
System task model – Cellphone – (*XML*)
System task model – Desktop – (*XML*)
System task model – PDA – (*XML*)
Abstract UI – cellphone (*XML*)
Abstract UI – desktop (*XML*)
Abstract UI Pda (*XML*)
Openwave ... Nokia
(*WML, XHTML Mobile, VoiceXML, ...*)
M/Media PC Graphical PC Low Res PC
(*XHTML, ...*)
Palm ... Compaq

Figure 2: The transformations supported.

In the following sections we better explain such steps while showing their application to a specific example.

We have defined XML versions of the language for task modelling (ConcurTaskTrees), enabled task sets and the language for modelling abstract interfaces and developed automatic transformations among these representations.

4. TASK AND MULTI-PLATFORMS RELATIONSHIPS

In general, when a multi-platform application is considered, it is important to understand what type of tasks can actually be performed in each available platform. We have identified a number of possibilities:

- *The same task can be performed on multiple platforms in the same manner* (there may be only some changes in attributes of the user interface objects from platform to platform). This is the case of tasks whose presentation remains mostly unchanged on different platforms: an example is when in a museum application textual links are provided to access general information about the museum (how to reach, timetable, etc.).
- *Same task on multiple platforms but with different user interface objects.* An example of this case is highlighted in figure 3. In both systems users can select a section of the museum (e.g. Roman Archaeology, Modern Sculpture, etc). However, while in the desktop system a large, coloured interactive map of the museum is at the users' disposal, in the phone, because of its limited capabilities, a text link is available for every museum section.

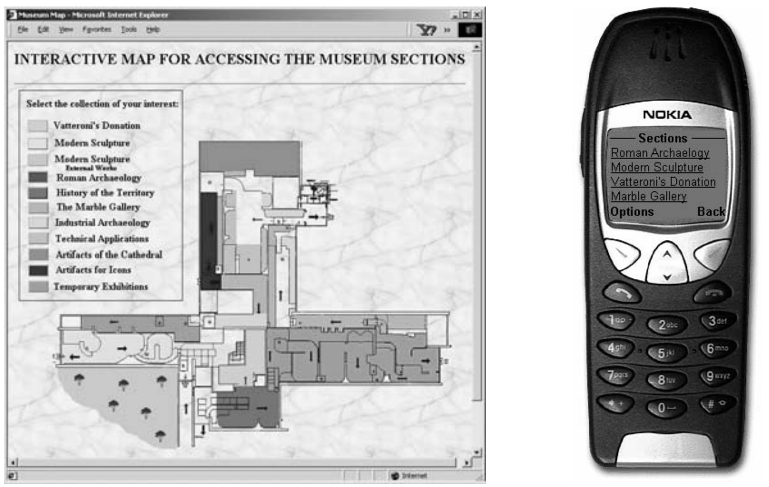
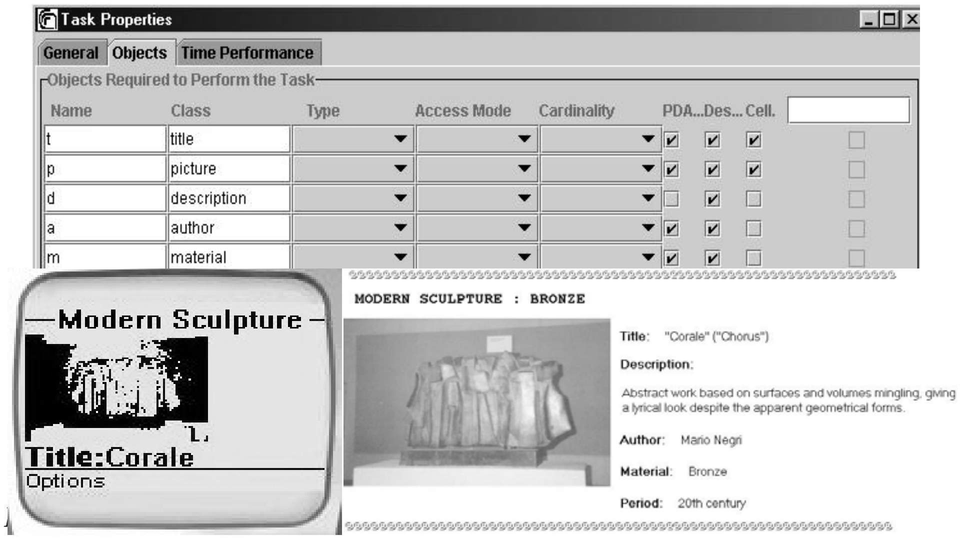


Figure 3: Same task, different interface objects

- *Same task on multiple platforms but with different domain objects*. This means that during the performance of the same task different sets of domain objects are manipulated. Figure 4 shows an example of this: presentations of different information on a desktop system and a WAP phone. As you can see from the bottom part of the picture, while in the desktop system it is possible to access a wider set of domain elements (title, image type, description, author, material, and date of creation), the WAP interface supports access to only an image (which is a low resolution image just to give users a rough idea of what the work of art is), along with the indications of the title and associated museum section. The top part of Figure 4 shows how in the CTTE tool it is possible to specify for each platform (PDA, desktop system, cellphone or other platforms) what objects can be manipulated during performance of the task in question. For example, for the "*Show artwork info*" task considered in the picture, the designer has specified that the title is available on all the platforms considered (all the checkboxes labelled "*PDA*", "*Desktop*", and "*Cellphone*" have been selected), whereas the description is available only for the desktop system. This information has been used to generate different user interfaces for the cellular phone and the desktop system.



Objects Required to Perform the Task

Name	Class	Type	Access Mode	Cardinality	PDA...	Des...	Cell.	
t	title				✓	✓	✓	☐
p	picture				✓	✓	✓	☐
d	description				☐	✓	☐	☐
a	author				✓	✓	☐	☐
m	material				✓	✓	☐	☐

Modern Sculpture

Title:Corale

Options

MODERN SCULPTURE : BRONZE

Title: "Corale" ("Chorus")

Description:

Abstract work based on surfaces and volumes mingling, giving a lyrical look despite the apparent geometrical forms.

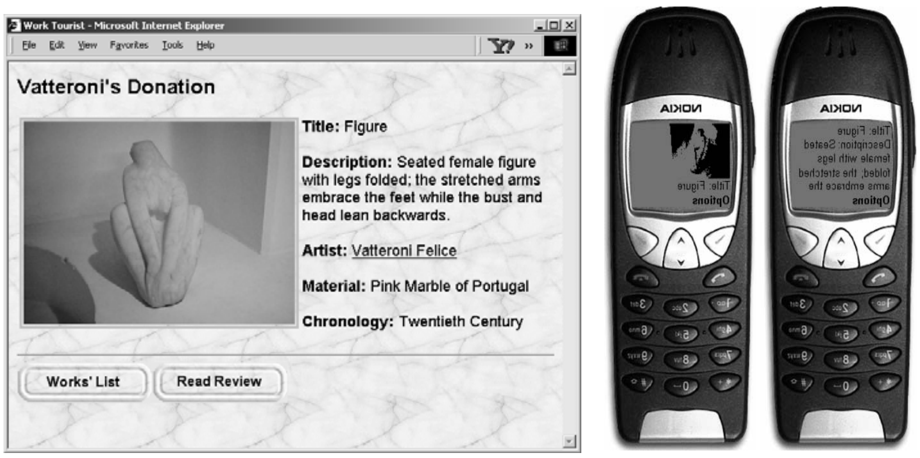
Author: Mario Negri

Material: Bronze

Period: 20th century

Figure 4: Same task, different domain objects

- *Same task on multiple platforms but with different task decomposition*. This means that the task is sub-divided differently, with different sets of sub-tasks, depending on the platform. An example of this possibility is displayed in the figure 5 that shows how differently the task *access work of art* is supported in a desktop and in a wap device. In the desktop system, the users can accomplish additional subtasks, which are not supported in other systems. An example concerns the possibility of reading reviews of a particular work of art, which is a lengthy information-processing task that users can perform satisfactorily when sitting in front of a desktop computer, but which is simply unacceptable with handheld devices.


Work Tourist - Microsoft Internet Explorer

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.

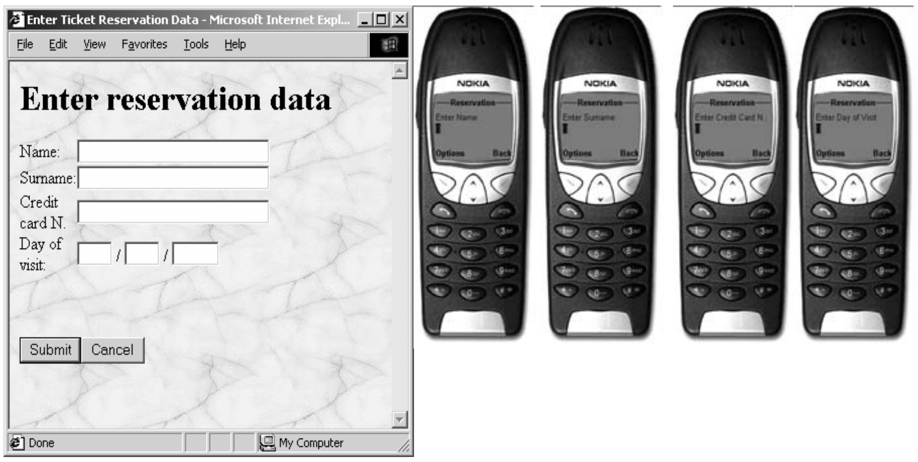
Artist: Vatteroni Felice

Material: Pink Marble of Portugal

Chronology: Twentieth Century

Figure 5: Same task, different task structure

- *Same task on multiple platforms but with different temporal constraints*. In this case the difference is in the temporal relationships among the subtasks. With reference to the museum application, consider the case of users who wish to electronically reserve their tickets for a particular visit in order to avoid queues. As you can see from the picture, in both systems they have to provide personal information. However, while in the desktop system they are free to choose the order to follow for filling in the various fields, within the phone application they are constrained by the wap interface to follow a sequential order (see Figure 6).

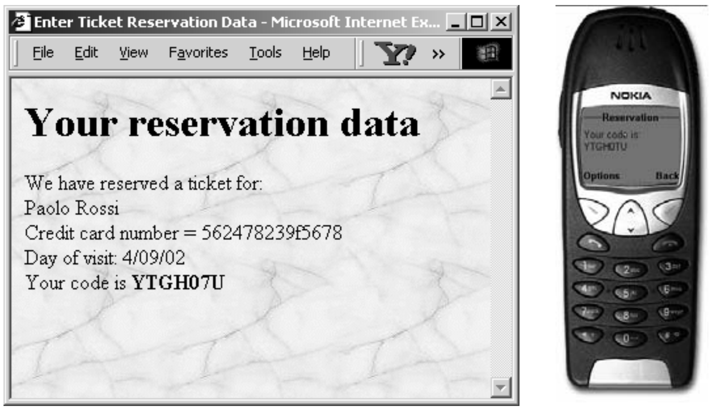

Enter Ticket Reservation Data - Microsoft Internet Expl...

Enter reservation data

Name:	
Surname:	
Credit card N.	
Day of visit:	/ /

Figure 6: Same task, different temporal relationships

- *Dependencies among tasks performed on different platforms*. An example of this can be found when the users have to reserve their tickets. Through the desktop system users can access, compare and contrast the different options about the best time for visiting the museum (depending on planned exhibitions, time, etc.). Once they have selected their preferences and entered personal data, the system provide them with a special reservation code identifying the data associated to that specific visit. It is only when they arrive at the museum that users need such reservation code, and the WAP device they usually bring should be able to show such information necessary to pick up the ticket at the museum and avoid queuing (see Figure 7). Capturing this type of task relationships is particularly important when there is some task relevant to only a particular platform and that affects the performance of another task through a different platform. A typical situation occurs when users physically visit the museum and simultaneously annotate the most interesting works of art on the PDA. When they arrive home they would appreciate being able to receive information regarding such works first during their access to the museum web site through a desktop system.


Enter Ticket Reservation Data - Microsoft Internet Ex...
File Edit View Favorites Tools Help
NOKIA
Options
Back

Your reservation data

We have reserved a ticket for:
Paolo Rossi
Credit card number = 562478239f5678
Day of visit: 4/09/02
Your code is **YTGH07U**

Reservation
Your code is:
YTGH07U

Figure 7: Relationships among tasks performed on different platforms

5. FROM THE TASK MODEL TO THE ABSTRACT USER INTERFACE

Starting with the task model of the system, we aim to identify the specification of the abstract user interface in terms of its static structure (the "*presentation*" part) and dynamic behaviour (the "*dialogue*" part): such abstract specification will be used to drive the implementation. By analysing the temporal relationships of a task model, it is possible to identify the sets of tasks that are enabled over the same period of time according to the constraints indicated in the model (*enabled task sets*). Thus, the interaction techniques supporting the tasks belonging to the same enabled task set are logically candidates to be part of the same presentation, though this criteria should not be interpreted too rigidly in order to avoid excessively modal user interfaces.

This shift from task to abstract interaction objects is performed through three steps:

- *Calculation of enabled task sets*; we have developed an algorithm that takes as input the formal semantics of the temporal operators of the ConcurTaskTrees notation and the specification of a task model and identifies the corresponding enable task sets;
- *Heuristics for optimisation in terms of presentation sets and transitions*; since a direct mapping between enabled task sets and user interface presentations can generate excessively modal user interfaces or interfaces with a very limited number of elements, these heuristics help designers to group tasks in presentation sets that are better candidates to support the mapping into the user interface presentations.
- *Mapping presentation task sets and their transitions into sets of abstract interaction objects and dialogue.*

5.1 Identification of Presentation Task Sets

The first step is to calculate the Enabled Task Sets (ETSs) according to the system task model. The CTTE tool automatically performs the identification of these sets. Only application and interaction tasks are considered in ETSs because user tasks (those associated with internal cognitive activities) are not directly relevant to this transformation. The ETSs identify a number of potential presentations and the connections among different ETSs are represented by transition tasks.

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 Sets* or PSs. The reasons for such step are various: first of all, reducing the initial number of ETSs which —as we previously noted— 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 which 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:

- *H1*: If two (or more) ETSs differ for only one element, and those elements are at the same level connected with an enabling operator, they could be joined together.
- *H2*: If an ETS is composed of just one element, it should be joined with another ETS that shares some semantic feature.
- *H3*: If some ETSs share most elements, they could be unified. For example if the common elements all appear at the right of the disabling operator, they could be joined in only one PS.
- *H4*: If there is an exchange of information between two tasks, they can be put in the same PS in order to highlight such data transfer.

It is worth noting that it is the designer that decides 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 applied less 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.

5.2 The Language for Abstract User Interfaces

The set of PSs obtained is the initial input for building the abstract user interface specification, which will be composed of interactors (Paternò and Leonardi, 1994) (abstract interaction objects) associated with the basic tasks. Such interactors are high-level interaction objects that are classified first de-

pending on the type of task supported, then depending on type and cardinality of the associated objects and lastly on presentation aspects.
Figure 8 provides a tree-like representation of the abstract language that has been used for specifying the abstract user interface. As you can see from the picture, an interface is composed of one or more presentations and each presentation is characterised by a *structure* and 0 or more *connections*. The basic idea is that the *structure* describes the static organisation of the user interface, whereas the connections describe the relationships among the various presentations of the user interface. Generally speaking, the set of connections identifies how the user interface evolves over time, namely its dynamic behaviour.

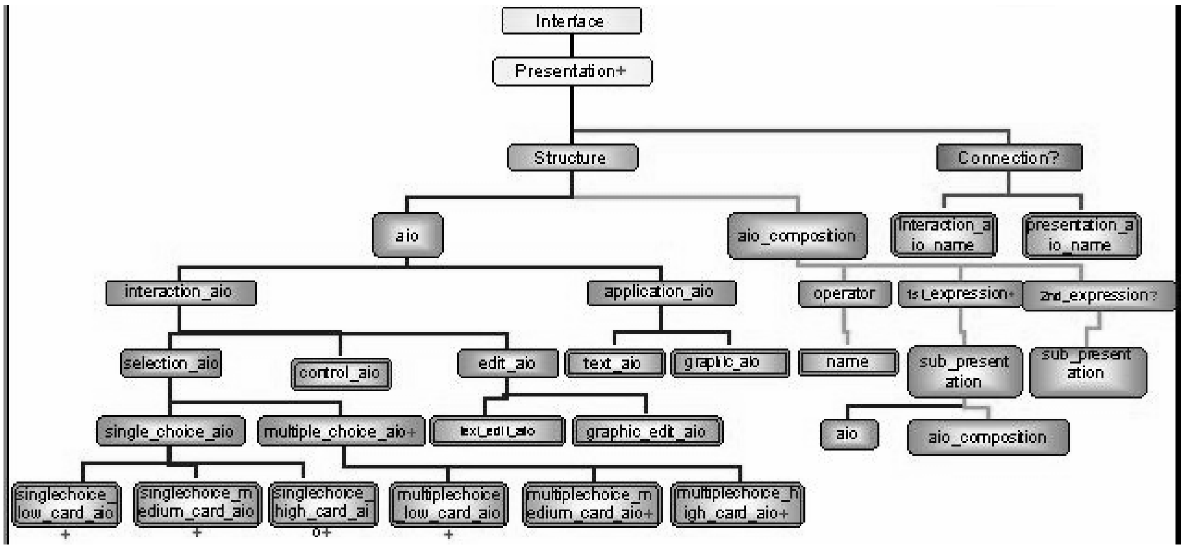


Figure 8: The structure of the abstract user interface language

In fact, each connection has two attributes: an *interaction_aio_name*, which defines the interaction object whose performance triggers the next presentation, which is identified by the *presentation_aio_name.*

As far as the *structure* is concerned, two types of elements can occur: elementary abstract interaction objects (*aio*), or complex expressions (*aio_composition*) derived from applying the *operators* to such objects. Each aio can be either an *interaction_aio* or an *applicaton_aio* depending on whether or not an interaction between the user and the application is involved. For *interaction_aio* we have various categories depending on the type of basic task supported (editing, selection, etc.). For the *application_aio* we have different classes according to the types of object manipulated.

5.3 From Presentation Task Sets to Abstract User Interface Presentations

The abstract user interface is mainly defined by a set of interactors and the associated composition operators.

The type of task supported, the type of objects manipulated and their cardinality are useful elements for identifying the interactors. In order to compose such interactors we have identified a number of composition operators that capture typical effects that user interface designers actually aim to achieve (Mullet and Sano, 1995):

- ***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 unrelevant.
- ***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. The importance can be derived from the frequency of access or depend on the application domain. 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'.

At this point we have to map each task of the presentation set considered into a suitable interactor and build a presentation structure where the relationships among tasks are reflected through the different relationships between such interactors which are expressed by using the composition operators. In order to derive the presentation structure associated to the specific presentation set and deduce the operators that should be applied to them, we have to consider the part of the task model regarding the tasks belonging to a specific presentation set. In this process we have to consider that temporal relationships existing between tasks are inherited also by their subtasks.

5.4 The Dialogue Part

Once the *static* arrangement of the abstract user interface is identified, we have to specify its *dynamic* behaviour. To this aim, an important role is played by the so-called *transition tasks*. For each presentation set P, we define *transition tasks(P)* the tasks whose execution makes the abstract user interface pass from the current presentation set P into another presentation set P'. For each presentation set P, a set of rules (*transition_task*, *next_PS*) should be provided whose meaning is: when *transition_task* is executed, the abstract user interface passes from P to *next_PS*.

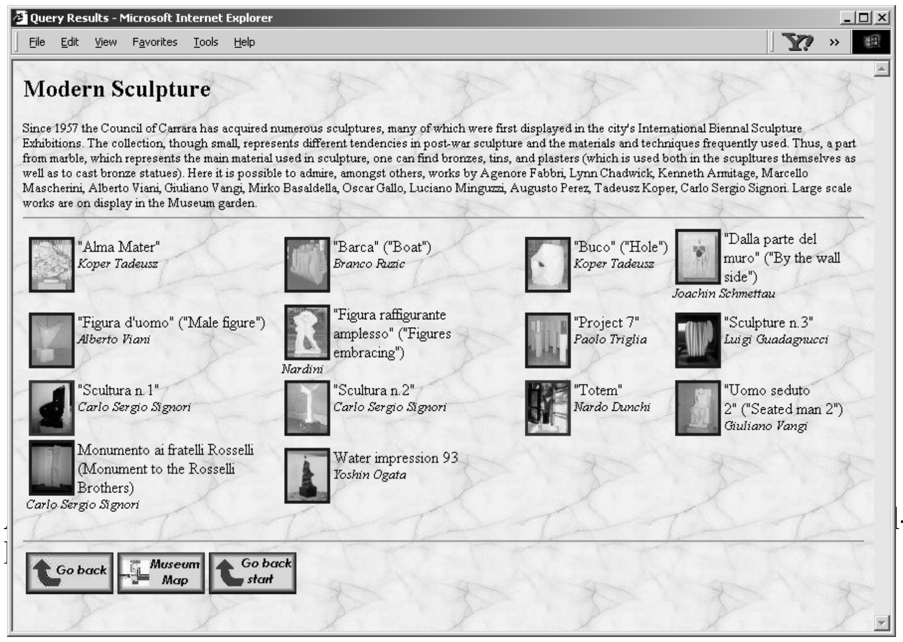
6. FROM THE ABSTRACT USER INTERFACE TO ITS IMPLEMENTATION

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 (operating system, toolkit, etc.), 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 (Mullet and Sano, 1995).

It is worth noting that the composition operators can be iteratively applied on the elements of the domain objects, and we refer to this possibility by putting a * beside the name of the operator (e.g. G*, H*). This means that each interactor involved in the expression actually involves multiple user interface objects, so the operator is iteratively applied to every data element handled by the interactor in the expression. One example of this can be seen in the following expression, for the desktop system:

R(**H**(**G***(Show_work_info, show_work_image), Section_Description)), **G** (Go_back, Go_museum_map, Go_to_start))

This example shows an expression in which different operators have been used. Reading the expression, we have a grouping operator which involves three elements, namely the interactors associated to *Go_back, Go_museum_map and Go_to_Start* tasks respectively. This grouping is the right member of a Relation operator, which means that each of the elements appearing on the right side is in relation 1:N with the elements appearing on the left side of the R operator.
In turn, the left operand of the R operator is a complex expression which involves a Hierarchy operator. Recalling that H conveys the idea of hierarchy existing amongst the different objects involved, in this case the elements hierarchically arranged are a complex interactor expression (the iterated grouping of *Show_work_info, show_work_image*) and a basic interactor (that associated to the *Section Description* task). The latter interactor is considered less important than the first one and for this reason a smaller space has been allotted to it within the user interface (see Figure 9).


Query Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help

Modern Sculpture

Since 1957 the Council of Carrara has acquired numerous sculptures, many of which were first displayed in the city's International Biennal Sculpture Exhibitions. The collection, though small, represents different tendencies in post-war sculpture and the materials and techniques frequently used. Thus, a part from marble, which represents the main material used in sculpture, one can find bronzes, tins, and plasters (which is used both in the scupltures themselves as well as to cast bronze statues). Here it is possible to admire, amongst others, works by Agenore Fabbri, Lynn Chadwick, Kenneth Armitage, Marcello Mascherini, Alberto Viani, Giuliano Vangi, Mirko Basaldella, Oscar Gallo, Luciano Minguzzi, Augusto Perez, Tadeusz Koper, Carlo Sergio Signori. Large scale works are on display in the Museum garden.

"Alma Mater" *Koper Tadeusz*
"Barca" ("Boat") *Branco Ruzic*
"Buco" ("Hole") *Koper Tadeusz*
"Dalla parte del muro" ("By the wall side") *Joachin Schmettau*
"Figura d'uomo" ("Male figure") *Alberto Viani*
"Figura raffigurante amplesso" ("Figures embracing") *Nardini*
"Project 7" *Paolo Triglia*
"Sculpture n.3" *Luigi Guadagnucci*
"Scultura n.1" *Carlo Sergio Signori*
"Scultura n.2" *Carlo Sergio Signori*
"Totem" *Nardo Dunchi*
"Uomo seduto 2" ("Seated man 2") *Giuliano Vangi*
Monumento ai fratelli Rosselli (Monument to the Rosselli Brothers) *Carlo Sergio Signori*
Water impression 93 *Yoshin Ogata*

Figure 9: Example of implementation on desktop system

Another example of application can be seen by considering the following expression:

H(Show section list, **G** *****(Access to section info, Access to section works))

This expression has, at its outermost level a Hierarchy operator, whose first element ('first' according to the hierarchy as well) is the interactor associated to the *Access to sections' list* task. The right-hand operand is an iterated grouping of the interactors associates to *Access to section's info* and to *Access to section's works* tasks. In figure 10 it is shown how this expression can be rendered on a cellphone user interface, where there are less possibilities due to the more limited capabilities of the device. As you can see the interactor associated to *Access to sections' list* task has more predominance with respect to the grouped expression and this is conveyed by using a bigger size of the font for displaying the related textual string.

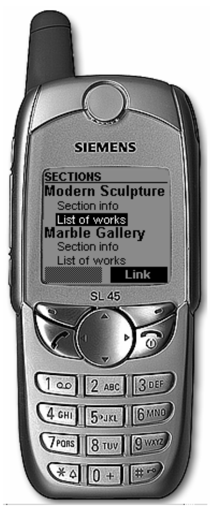


Figure 10: Example of implementation on a Wap device

7. AN EXAMPLE

The example considered throughout the paper regards the design of a nomadic application for accessing museum information. Its task model contains tasks that refer to the possible platforms according to the various possibilities discussed in section 4. There are tasks performed through only one platform (such as the possibility of accessing the museum map, namely *Access to map* task,), tasks that are performed differently according to the type of platform (such as *Show artwork info*) and so on.

A possible scenario is a user who comes across the museum while surfing the web at home. S/he accesses the web site and starts to look at the information contained and afterwards s/he proposes a visit to the Marble Museum to some friends. As they appear a bit reluctant, s/he accesses the WAP server trying to convince them by showing some concrete elements of what the museum offers. Finally, they decide to have a visit, so they access again the WAP server to check its location and schedule. When they arrive at the museum, they receive a guide implemented in PDA that provides audio-visual support for their visit.

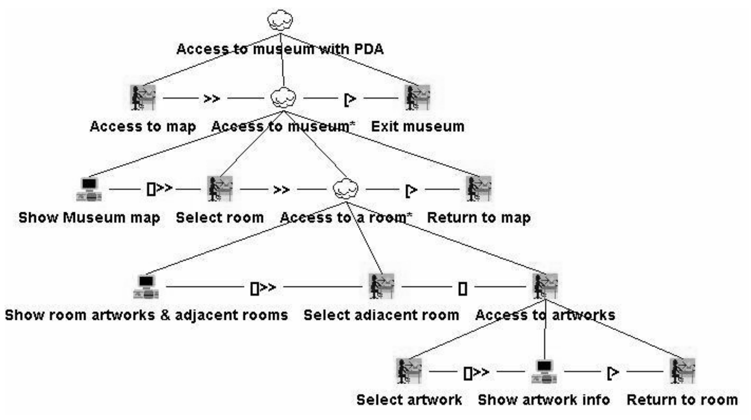


Figure 11: Part of the system task model for the PDA access

For reasons of space we have provided only part of the system task model for the PDA (see Figure 11). Users can access the museum map (*Access to map*), and then (see the enabling CTT operator ">>"), after such map is displayed (*Show museum map*), they can select a room (*Select room*). When they select a room, the presentation shows the different types of artworks in the room, so as to allow visitors to locate them. After the visitor selects a specific artwork (*Select artwork*), depending on the selection (see the enabling with information passing operator ”[]>>”) the PDA delivers information about the artwork (*Show artwork info*), with the possibility of returning to the previous room. Further actions will be made available to the user: either return to the global map (*Return to map*) or exit the museum (*Exit museum*), both appearing at the right of a disabling operator (“[>”).

For the considered example, the ETSs are:
ETS1: {Access to map}
ETS2: {Show Museum map, Exit museum}
ETS3: {Select room, Exit museum}
ETS4: {Show room artworks&adjacent rooms,Return to map, Exit museum}
ETS5: {Select adjacent room, Select artwork, Return to map, Exit museum}
ETS6: {Show artwork info, Return to room, Return to map, Exit museum}

By applying the heuristics we may find for example that ETS2 and ETS3 differ by only one element: applying H1 they could be unified into *PS1*={*Show museum map*, *Select room*, *Exit Museum*}. Also, ETS4 and ETS5 share most elements: H3 can be applied to obtain *PS2*={*Select adjacent room*, *Select artwork*, *Show Room Artworks & Adjacent Rooms*, *Return to map*, *Exit museum*}.

For example, the presentation structure then obtained for PS2 is:
O (*Show Room Artworks & Adjacent Rooms*, **G** (*Select adjacent room*, *Select artwork*)) **R** *Return to map* **R** *Exit museum*
This is because the Ordering operator highlights the information transfer (specified by the []>> operator) between the *Show Room Artworks & Adjacent Rooms* and both *Select adjacent room* and *Select artwork* tasks (which are grouped together because of the choice “[]” operator). Each task is in turn put in relation to *Return to map* and *Exit museum* because they both appear at the right of a disabling operator ([>). In the same way, we can identify the abstract presentation associated to PS1 which is: **O**(*Show museum map*, *Select room*) **R** *Exit Museum*

Regarding the transitions, we can see, for example, that PS1 has two transition tasks: *Select Room* and the task *Exit Museum*. To express that via the transition task *Select room* the abstract interface passes from PS1 to PS2 we can use the following rule:
<*presentation_set* **PS1** /*presentation_set* ><*behaviour*><*rule*>
<*transition_task* **Select room** /*transition_task*> <*next_PS* **PS2** /*next_PS*>
</*rule*></*behavior*>

In addition, we have to map each task into a suitable interactor, by considering relevant dimensions specified in the task model. For example with regard to a *selection* task (as *Select adjacent room* task) we identify the *type of selection* (single/multiple), the *type* of manipulated objects (boolean/quantitative/text, etc) and the *cardinality* (low/medium/high) of the dataset from which the selection should be performed, as relevant dimensions. Once such attributes have been identified, we define some rules to indicate in which case a widget is more appropriate than another one depending on the different values that each attribute assumes and on the specific platform and/or device considered. For example, as the *Select adjacent room* task is single selection task managing spatial information and the set of manipulated objects has a low cardinality, the interaction technique to be provided must allow a choice of elements by providing an interactive map.

A possible implementation for the presentation corresponding to PS2 is shown in Figure 12, supposed that the current room is about Roman Archaeology.

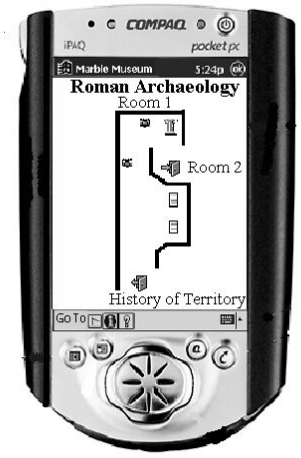


Figure 12: One presentation of the PDA user interface

8. CONCLUSIONS

In this paper we have described a method for designing multi-device, nomadic applications starting from their task model and using a number of transformations in the design cycle to obtain final applications able to effectively support the users' activities through various devices accessed in different contexts. The application of the method helps maintain a high-level of consistency across the multiple user interfaces developed.

We are developing a tool, TERESA (Transformation Environment for inteRactivE Systems representAtions), whose aim is to provide a complete semi-automatic environment supporting the presented method and transformations. Particular attention will be paid to design a high-level control panel for designers so that they can focus on main design aspects and choices through effective representations without even knowing the basic underlying mechanisms and concepts (such as enabled task sets) that support the possible transformations.

This work has been supported by the CAMELEON IST project http://giove.cnuce.cnr.it/cameleon.html. We thank the European Commission for the support and project colleagues for useful discussions on the topics of the paper.

REFERENCES

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

Beyer, H., & Holtzblatt, K., 1998. Contextual Design: Defining Customer Centred Systems. San Francisco: Morgan Kaufman.

Calvary, G., Coutaz, J., Thevenin, D., 2001. A Unifying Reference Framework for the Development of Plastic User Interfaces, Proceedings of EHCI 2001, Springer Verlag.

Einsenstein, J., Vanderdonckt, J., Puerta, A., 2001. Applying Model-Based Techniques to the Development of UIs for Mobile Computers, Proceedings of the Fifth International Conference on Intelligent User Interfaces (IUI '01), ACM Press.

Johnson, P., Wilson, S., Markopoulos, P., Pycok, J., 1993. ADEPT - Advanced Design Environment for Prototyping with Task Models, Proceedings of InterCHI'93, Amsterdam, The Netherlands, pp 56-57.

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

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

Olsen, D., 1998. Interacting in Chaos, Keynote address. Proceedings of IUI'98, San Francisco, pp.97-100.

Paternò, F., 1999. Model-based Design and Evaluation of Interactive Applications, Springer Verlag.

Paternò, F., Leonardi, A., 1994. A Semantics-based Approach to the Design and Implementation of Interaction Objects, Computer Graphics Forum, Blackwell Publisher, Vol.13, N.3, 195-204.

Puerta, A.R., 1997. A Model-Based Interface Development Environment, IEEE Software, July/August 1997, 40-47.

Sukaviriya, P.N., Foley, J.D., and Griffith, T., 1993. A second generation user interface design environment: The model and the runtime architecture, Proceedings of INTERCHI '93, Amsterdam. ACM Press, New York, NY, pp.375–382.

Szekely, P., Luo, P., and Neches, R., 1993. Beyond interface builders: Model-based interface tools, Proceedings of INTERCHI '93. ACM Press, New York, NY, pp.383–390.

Thevenin, D., Coutaz, J., 1999. Plasticity of User Interfaces: Framework and Research Agenda, Proceedings of Interact ‘99, Edinburgh, A. Sasse & C. Johnson Eds, IFIP IOS Press Publ., pp.110-117.

Vanderdonckt, J., Bodart, F., 1993. Encapsulating Knowledge For Intelligent Automatic Interaction Objects Selection, Proceedings of InterCHI'93, pp 424-429.