Available online at www.sciencedirect.com
SCIENCE DIRECT®
Interacting with Computers 17 (2005) 291–315
Interacting with Computers
www.elsevier.com/locate/intcom

# Model-based tools for pervasive usability

Fabio Paternò*

*ISTI-CNR, Via G.Moruzzi 1, Pisa 56124, Italy*

Received 21 January 2004; revised 22 June 2004; accepted 29 June 2004
Available online 15 September 2004

**Abstract**

This paper aims to provide a discussion of how model-based approaches and related tools have been used to address important issues for obtaining usable interactive software and the new challenges for this research area. The paper provides an analysis of the logical descriptions that can be used in the design of interactive systems and how they can be manipulated in order to obtain useful results. This type of approach has recently raised further interest in the ubiquitous computing field for supporting the design of multi-device interfaces. The new challenges currently considered are mainly in the area of end-user development, ambient intelligence, and multimodal interfaces.


*Keywords:* Model-based approaches; Tools; Usability; Task models; Multi-device interfaces

## 1. Introduction

Models can help in the design and evaluation of interactive applications even if such approaches have sometimes been criticised by researchers who have considered them too theoretical and a sort of useless complication. However, if we think of what we do in our daily practice we can discover that we often use models. As soon as there is some complex entity to manage, people try to identify the main aspects that should be taken into account and their relations. For example, when we wake up in the morning we often think about the main activities to perform, thus creating a model of the new day. So, we build models to understand reality and to guide our interactions with it. Such models can be represented with different levels of formality. Tools are important in order to ease their development,

* Tel.: +390-50-315-3066; fax: +390-50-313-8091.
*E-mail address:* fabio.paterno@isti.cnr.it.

0953-5438/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.intcom.2004.06.017

analysis and use in the design process. In the design of interactive systems the range of possible design choices is wide and has to consider many aspects. Model-based approaches can be helpful to manage such complexity.

The technological evolution underway has brought with it an ever increasing number of users with a greater availability of interaction platforms, working in the most varied environments. This tendency has raised further interest in model-based approaches. Even researchers in user interface tools who have often been sceptical regarding such approaches admit (Myers et al., 2000) that they can be useful for developers, vendors and users in order to reason about user interface design solutions, especially when multi-device applications are considered.

More generally, model-based approaches can provide useful support to address the challenges raised by pervasive computing. Models can provide a structured description of the relevant information, and then intelligent environments can render the associated elements taking into account the actual context of use. Pervasive usability means the ability of still supporting usability even when interactions can be performed under the 'anytime–anyhow–anywhere' paradigm. This means users are able to seamlessly access information and services regardless of the device they are using or their position, even when the system and/or the environment change dynamically.

The goal of this paper is to review work done so far in the area of model-based approaches and discuss the new challenges facing designers in this area. Particular attention is devoted to concepts, methods and tools that involve the use of task models in order to obtain usable interfaces. In the discussion, I use some results of my research work, as well as those achieved by other groups, to exemplify and explain the various approaches, though for the sake of brevity I cannot cite all of them. In the last fifteen years various model-based approaches have been proposed. An overview paper on model-based technologies was written in (Szekely, 1996). It is interesting to discuss how the field has evolved after some years and see how the challenges (task-centred interfaces, multi-platform support, interface tailoring, multi-modal interfaces) that were identified at that time have been addressed and what the new challenges are.

In the paper I first recall some basic concepts in this type of approach and some of the previous work in the area. Then, I focus on discussing current work in this area that centres on how model-based approaches can address the design of multi-device interfaces. The final part is dedicated to discussing some challenges for the coming years, such as their use for supporting ambient intelligence or the need for new evolutions in order to become an integral part of end user development.

### *1.1. Basic concepts*

As often happens, the solution to a complex problem, such as the design of an interactive system, can be based on a small set of clear basic concepts. In order to address such issues it is important to consider the various viewpoints that it is possible to have on an interactive system. Such viewpoints differ for the abstraction levels (to what extent the details are considered) and the focus (whether the task or the user interface is considered). The model-based community has long discussed such possible viewpoints, see for example (Szekely, 1996; Paternò, 2003). Such abstraction levels are:

- *Task and object model*, at this level, the logical activities that need to be performed in order to reach the users' goals are considered. Often they are represented hierarchically along with indications of the temporal relations among them and their associated attributes. The objects that have to be manipulated in order to perform tasks can be identified as well.
- *Abstract user interface*, in this case the focus shifts to the user interface supporting task performance. Only the logical structure is considered, in a modality-independent manner, thereby avoiding low-level details. Interaction objects are described in terms of their semantics through interactors (Paternò and Leonardi, 1994). Thus, it is possible to indicate, for example, that at a given point there is a need for a selection object without indicating whether the selection is performed graphically or vocally or through a gesture or some other modality.
- *Concrete user interface*, at this point each abstract interactor is replaced with a concrete interaction object that depends on the type of platform and media available and has a number of attributes that define more concretely how it should be perceived by the user.
- *Final user interface*, at this level the concrete interface is translated into an interface implemented by a specific software environment (e.g. XHTML, Java,…).

To better understand such abstraction levels we can consider an example of a task: making a hotel reservation. This task can be decomposed into selecting arrival and departure dates and other subtasks. At the abstract user interface level we need to identify the interaction objects needed to support such tasks. For example, for easily specifying arrival and departure days we need selection interaction objects. When we move on to the concrete user interface, we need to consider the specific interaction objects supported. So, in a desktop interface, selection can be supported by a graphical list object. This choice is more effective than others because the list supports a single selection from a potentially long list of elements. The final user interface is the result of these choices and others involving attributes such as the type and size of the font, the colours, and decoration images that, for example, can show the list in the form of a calendar.

Many transformations are possible among these four levels for each interaction platform considered: from higher level descriptions to more concrete ones or vice versa or between the same level of abstraction but for different type of platforms or even any combination of them. Consequently, a wide variety of situations can be addressed. More generally, the possibility of linking aspects related to user interface elements to more semantic aspects opens up the possibility of intelligent tools that can help in the design, evaluation and run-time execution.

Fig. 1 shows the models considered and how tools can exploit them in the development process. The context model provides declarative descriptions of environments, platforms, and users. There are tools that help in developing the models, others that aim to analyse their content and others that use them in order to generate the user interface. For this purpose these latter support set of design criteria and can implement transformations through different abstraction levels. Other possibilities are offered by tools that take the user interface implementation and reconstruct the corresponding models that can then be modified or analysed through the other tools.

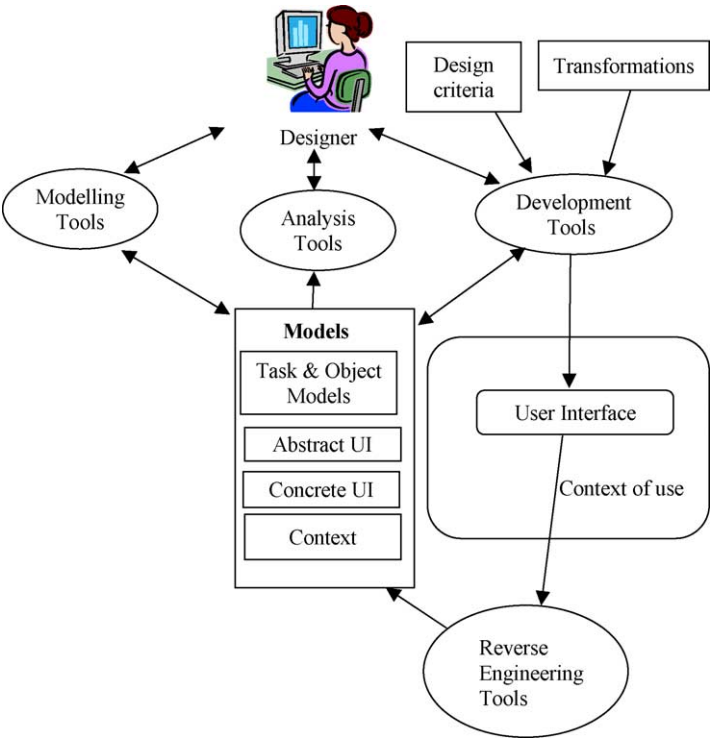


Fig. 1. Models and related tools in the user interface development process.

### *1.2. Model-based approaches in interactive systems*

The purpose of model-based design is to identify high-level models which allow designers to specify and analyse interactive software applications from a more semantic-oriented level rather than starting immediately to address the implementation level. This allows them to concentrate on more important aspects without being immediately confused by many implementation details and then to have tools which update the implementation in order to be consistent with high-level choices. Thus, by using models which capture semantically meaningful aspects, designers can more easily manage the increasing complexity of interactive applications and analyse them both during their development and when they have to be modified.

The first generation of model-based approaches used conceptual descriptions between the abstract and the concrete user interface levels. One of the first works in this area was the User Interface Development Environment (UIDE) (Foley and Sukaviriya, 1994) developed by Jim Foley's group at the GVU Center of Georgia Tech. In this environment it was possible to specify the pre- and post-conditions related to each interaction object of the user interface. Pre-conditions have to be satisfied in order to make the interaction object reactive whereas post-conditions indicate modifications of the state of the user interface which have to be performed after a user interaction with

the related object. Humanoid (Szekely et al., 1993) provided a declarative modelling language consisting of five independent parts: the application semantics, the presentation, the behaviour, the dialogue sequencing, and the action side effects. One of its purposes was to overcome one limitation of traditional tools for user interface development: lack of support for exploration of new design that requires undoing portions of the existing design and adding new code. The reason for this limitation was that such tools do not have a notion of what is specific to a design as this information is embedded in the code.

In the second generation of model-based approaches there was a general agreement that task models are important models in user interface design and the model-based proposals included some sort of task models. A task-driven approach was supported by Adept (Wilson et al., 1993). In their proposal they address the design of a task model, an abstract architectural model, and a related implementation. This was one of the first research prototypes aiming to use task models to support user interface development. However, the set of temporal operators among tasks considered and the rules for creating relationships within the levels considered have then been discussed and expanded in other methods and tools. Trident (Bodart et al., 1994) used an activity chaining graph to specify the information flow between the application domain functions and an entity-relationship diagram for the information modelling part of the functional requirements. The MOBI-D (model based interface designer) (Puerta and Eisenstein, 1999) approach is a model-based interface development environment for single-user interfaces that enables designers and developers to interactively create user interfaces by designing interface models. The environment integrates model-editing tools, task elicitation tools (Tam et al., 1998), an intelligent design assistant and interface building tools.

There are other types of models that can provide support for the design of an interactive application even if they have different main goals. On the one hand, in the computer science area, we can find more traditional object-oriented modelling techniques. The most successful has been the unified modelling language (UML) (Booch et al., 1999). They share similar purposes with task-oriented approaches. In both cases there is a description of both activities and objects. The main difference is in the focus and the notations used to represent the relevant concepts. Task-based approaches first identify activities and then the objects that they have to manipulate. Object-oriented methods follow an inverse process as they mainly focus on modelling the objects composing the system. Consequently, task-based approaches are more suitable to design user-oriented interactive applications because in this way the main focus is on effectively and efficiently supporting users' activities (one of the most important basic usability principle is 'focus on the user and their tasks') whereas object-oriented techniques have been more successful at engineering the software implementation level.

On the other hand, in the cognitive science field, various types of cognitive models have been proposed in the literature. In cognitive architectures there is an integrated description of how human-cognitive mechanisms interact with each other. This makes their simulation by automatic tools possible. Examples of cognitive architectures are ICS (interacting cognitive subsystems) (Barnard and May, 1995) or EPIC (Kieras, 1996).

## 2. Task modelling

Of the relevant models in the human–computer interaction field, task models play an important role because they represent the logical activities that should support users in reaching their goals. Knowing the tasks necessary to goal attainment is fundamental to the design process. The need for modelling is most acutely felt when the design aims to support system implementation as well. If we gave developers only informal representations (such as scenarios or paper mock-ups), they would have to make many design decisions on their own, likely without the necessary background, to obtain a complete interactive system. Task models represent the intersection between user interface design and more systematic approaches by providing designers with a means of representing and manipulating an abstraction of activities that should be performed to reach user goals.

It is important to distinguish between the task analysis and the task modelling phases. The purpose of task analysis is to identify what the relevant tasks are. This understanding not only requires a strong end user involvement but must also take into account how activities are performed currently. The task modelling phase occurs after the task analysis phase. The purpose of task modelling is to build a model which describes precisely the relationships among the various tasks identified. These relationships can be of various types, such as temporal and semantic relationships.

In order to be meaningful, even the task model of a new application should be developed through an interdisciplinary collaborative effort involving the various relevant viewpoints. By drawing on the various fields of competence we can obtain user interfaces that can effectively support the desired activities. This means that the user task model (how users think that the activities should be performed) shows a close correspondence to the system task model (how the application assumes that activities are performed).

There are many reasons for developing task models. In some cases the task model of an existing system is created in order to better understand the underlying design and analyse its potential limitations and how to overcome them. In other cases designers create the task model of a new application yet to be developed. In this case, the purpose is to indicate how activities should be performed in order to obtain a new, usable system that is supported by some new technology.

Task models can be represented at various abstraction levels. When designers want to specify only requirements regarding how activities should be performed, they consider only the main high-level tasks. On the other hand, when designers aim to provide precise design indications then the activities are represented at a small granularity, thus including aspects related to the dialogue model of a user interface (which defines how system and user actions can be sequenced).

The subject of a task model can be either an entire application or one of its parts. The application can be either a complete, running interactive system or a prototype under development. The larger the set of functionalities considered, the more difficult the modelling work. Tools such as CTTE (Mori et al., 2002) (publicly available at http://giove.isti.cnr.it/ctte.html) open up the possibility of modelling entire applications, but in the majority of cases what designers wish to do is to model some sub-sets in order to analyse them, and to identify potential design options and better solutions.

Task models can be used for many types of applications: there are applications that are clearly goal-oriented and so the tasks are clearly structured, other applications support a wide range of options open at any time, with the continuous possibility for the users to freely decide what to do and how to do it, in this case the task model should not be particularly structured in order to allow such possibilities.

### 2.1. Discussion of task modelling approaches

It can be useful to consider some of the most common criticisms of task modelling approaches in order to better understand their possibilities. Some of the most common issues (in italics) are cited and discussed in the following.

*The benefits of adopting task modelling do not justify the extra time required by their development.*

Designers have to think about the tasks to support and how the interface should actually support them. In some cases this is just a mental exercise, in other cases it is explicitly expressed through a notation for task modelling. The choice heavily depends on the size of the project and the application domain. It is clear that if a designer has to develop a small Web site, then the modelling is mainly a mental exercise and there is no particular need to express it through a notation and a tool. Vice versa, if the goal is to develop a large application with many people involved or the application design has to be periodically revised or the application domain is a safety-critical system, then there is a strong need for clarifying the design choices and documenting them through explicit task modelling. One obstacle to the explicit adoption of task modelling has been the lack of environments that facilitate this process. Task modelling is a useful exercise because it can stimulate discussion of design solutions among the different people involved (designers, developers, managers, end users and so on). Moreover, task models provide semantic information that can be useful for obtaining more effective context-dependent interfaces than other approaches.

*Task models are not applicable to creative tasks. They best support functionality where the user wants to achieve a certain goal through tasks whose relations can be modelled. Activities that do not directly 'lead anywhere' may be hard to describe.*

Task models allow designers to describe activities required to reach users' goals. The structure of the model should reflect the structure of the activities that designers want to enable. There are applications that are clearly goal-oriented and so the tasks are structured in-depth, there are other applications where people prefer to have a wide range of options at anytime and freely decide what to do and how to do it; in this case the task model should not be structured at all in order to allow for such possibilities.

*If the task hierarchy is implemented directly to the interface then every branch of this hierarchy is a mode.*

Hierarchical task models do not imply modes. The modes are generated by some type of temporal operators among tasks. So, for example, if tasks are concurrent, then there is no mode at all even with hierarchical tasks. A high level task is decomposed into a number of lower level ones that represent different options or different tasks at the same level that have to be performed to accomplish the higher level task. When there are multiple

hierarchies, then the temporal operators are used to describe their mutual relations. Some temporal operators can lead to modal dialogues, others to completely free and unstructured behaviours.

*Some designers prefer noun–verb style interaction over verb–noun style interaction. Task-based approaches lead to verb–noun style interfaces.*

The distinction between noun–verb styles versus verb–noun styles implies two different ways to accomplish tasks. In one case users first select a user interface object and then indicate what to do with it, in the other case users first select the activity that they aim to perform and then indicate what object it has to be applied to. So, task models can be applied equally to both interaction paradigms.

### 2.2. How task models can be represented

Many proposals have been put forward to represent task models. Hierarchical task analysis (Annett and Duncan, 1967) has a long history and is still cited and practiced, even more often than some more recent approaches. The concept of hierarchical decomposition of the activities to describe has shown to be successful because it allows designers to consider the various possible abstraction levels while still maintaining a clear indication of the relationships among them. Such hierarchical structure can be organised and presented in different ways (see Fig. 1): GTA (van Welie et al., 1998) expands it from left to right whereas the others expand it from top to down. Some of them allow only one temporal operator among the subtasks whereas others allow the possibility of using various operators among different subtasks. ConcurTaskTrees (CTT) (Paternò, 1999) also uses icons to visually present how the task performance is allocated. Some notations associate each node in the hierarchical structure with a task whereas others associate the node with a temporal operators and the leaves are associated with tasks (Violante 2001).

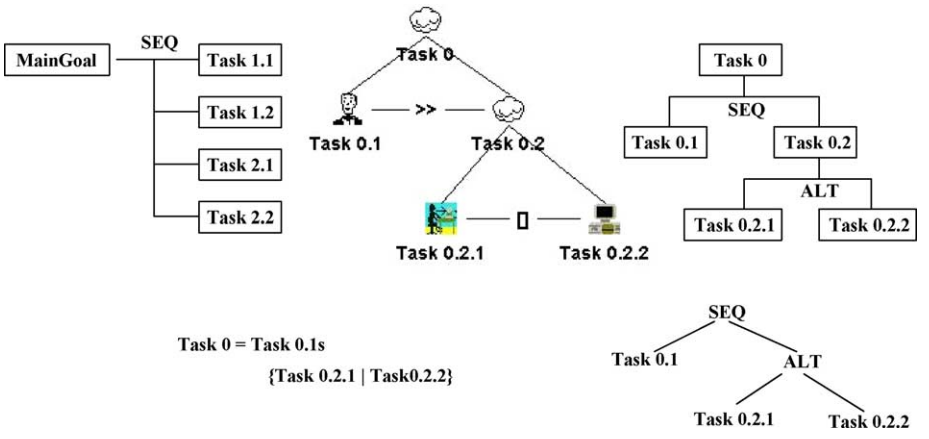


Fig. 2. Graphical notations for task modelling.

More generally, notations for task models can vary according to various dimensions (Fig. 2).

- *syntax (textual vs graphical)*, there are notations that are mainly textual, such as UAN where there is a textual composition of tasks enhanced with tables associated with the basic tasks. GOMS (John and Kieras, 1996) is mainly textual. ConcurTaskTrees and GTA, are mainly graphical representations aimed at better highlighting the hierarchical structure.
- *set of operators for task composition*, this is a point where there are substantial differences among the proposed notations. While GOMS supports only sequential tasks (with the exception of CPM-GOMS that also supports parallel tasks through the use of PERT-charts), UAN (Hartson and Gray, 1992) and CTT provide a much wider set of temporal relationships. This allows designers to describe more flexible ways to perform tasks.
- *level of formality*, the definition of a notation can be made in various manners. A formal definition can require time and be difficult to analyse; its advantage is that it provides precise meaning to its elements and how they can be composed. In some cases notations have been proposed aiming to provide a more intuitive representation of the concepts considered, in other cases more formal definitions have been proposed.

### 2.3. How tools can support the development of task models

Often it is difficult to create a model from scratch. To overcome this problem various approaches have been explored. CRITIQUE (Hudson et al., 1999) is a tool that aims to create KLM/GOMS models from the analysis of logs of user interactions with graphical interfaces implemented with a research tool. The model is created following two types of rules: the interaction operators are identified according to the type of event, and new mental operators are inserted when users begin working with a new object or when they change the input to the current object (for example, switching from clicking to typing in a text box). In this approach the limitation is that the task model only reflects the past use of the system and not other potential uses. These rules for building KLM/GOMS models including mental operators have then been used in another tool (John et al., 2004) that uses logs of interactions with Web pages. The authors reported that this approach was tested with two users (one was an author) and the models obtained were more accurate than those obtained by previously published approaches.

U-Tel (Tam et al., 1998) analyses textual descriptions of the activities to support and then automatically associates tasks with verbs and objects with nouns. CTTE provides the possibility of loading an informal textual description of a scenario or a use case and interactively selecting the information of interest for the modelling work. In this way, the designer can first identify tasks, then create a logical hierarchical structure and finally complete the task model.

The developers of ISODE (Paris et al., 2001) have considered the success of UML and provide some support to import use cases created by Rational Rose in their tool for task

modelling. This environment also includes TAMOT a tool for modelling tasks specified with the DIANE+ notation.

### 2.4. Task models analysis

The structure and the information of a task model can contain useful information for designers. Their analysis can be supported by tools in various manners, such as metrics evaluation and interactive simulation. CTTE supports the identification of a number of metrics for the analysis of a task model. This can be useful when designers want to compare how people work in the current system and how they could work in a new envisioned system or are interested in comparing the implications at task level of two alternative designs. The comparison is performed in terms of number of tasks, number of basic tasks (the tasks that are no longer decomposed), allocation of tasks, number of instances of temporal operators, structure of the task models (number of levels, maximum number of sibling tasks). This information can also be given for single task models in order to analyse them. By comparing this type of information it is possible to deduce some general feature of a solution with respect to another one. For example, a higher number of application tasks and a lower number of user tasks imply that there is a strong shift towards allocating task performance to the system, or a higher number of sequential operators implies that the solution supports a higher number of modes in its dialogues with the user. In Euterpe designers can specify constraints and heuristics. Constraints apply to every specification and should ideally have zero results. Heuristics can be used to analyse a specification, to find inconsistencies or problems. In this context, examples of heuristics that can be queried on the specification are: what tasks involve a certain role, what tasks occur more than a certain number of times, what tasks have more than a certain number of subtasks, what tasks have more than a certain number of levels.

A simulator for task models can be useful to better analyse the dynamic behaviour of task models, including those for cooperative applications. This feature is particularly meaningful when the notation used to represent the model allows the specification of many temporal relationships among tasks in addition to sequential tasks (such as disabling tasks, concurrent tasks, suspending tasks, and so on). This is a support that only a few tools provide (see for example VTMB (Biere et al., 1999) and CTTE). Also in the case of tools for UML this is a feature usually missing. When analysing an existent application or designing a new one it can be rather difficult for the designer to understand the dynamic behaviour resulting from the temporal relationships specified in the task model. The reason is that, especially for real applications, the number of ways in which the application can evolve is high and it is difficult to mentally remember the various temporal constraints among tasks and their possible effects. It becomes important to support a what-if analysis aiming at identifying what tasks are logically enabled if one specific task is performed. In interactive simulators the basic idea is that at any time they show the list of enabled tasks, according to the constraints specified in the task model. At any time, it is possible to go back through the performance of the tasks which means that the effect of the performance of the last task are undone and the list of enabled tasks becomes the same as that previous to the performance of the last task. At this point, the user can choose to go further backward in the task sequence or forward, either through the same path or a different one.

In summary, various solutions are possible for analysis tools based on task modelling. CTTE represents a useful contribution to understanding the possibilities in terms of the analyses that can be performed. We have seen that CTTE is also able to compare two models with respect to a set of metrics. Euterpe also supports the calculation of some metrics to analyse a specification, and help find inconsistencies or problems. The ability to predict task performance is usually supported by tools for GOMS such as QDGOMS (Beard et al., 1996). Overall, one important feature is the possibility of interactively simulating the task model's dynamic behaviour.

## 3. Current issues in model-based approaches

Recent years have seen the ever-increasing introduction of new types of interactive devices. A wide variety of new interactive platforms are offered on the mass market. A platform is a class of devices that share the same characteristics, especially in terms of interaction resources. They range from small devices such as interactive watches to very large flat displays. Examples of platforms are the desktop, the PDA, the mobile phone. The availability of such platforms has forced designers to strive to make applications run on a wide spectrum of platforms in order to enable users to seamlessly access information and services regardless of the device they are using and even when the system or the environment changes dynamically. If, on the one hand, this resulted in a dramatic improvement for the activities of users, on the other hand it has radically changed the nature of many interactive applications, converting them to nomadic applications, namely applications supporting user access in various contexts through different interactive devices. Thus, in order to guarantee a high level of user satisfaction it is necessary that the applications should be able to adapt their user interfaces to the different context of uses, in particular to the different devices used to access their functionality. This gives rise to the fundamental issue of how to assist software designers and developers in the development of interactive software systems able to adapt to different targets while preserving usability.

Currently the main issue addressed in model-based design is how to support design of multi-device interfaces. The tools addressing it can be considered the third generation of model-based tools. In this area we can identify three main topics: authoring environments that support editing of models and then transformations into the corresponding multi-device user interface implementations, tools that support reverse engineering of user interfaces in order to obtain the corresponding models, and XML languages for the various models that ease the manipulation through, even different, tools.

### *3.1. Authoring environments for model-based design*

MOBI-D is a particularly comprehensive tool because it provides a set of model editors (task, domain, presentation, user, and dialog) to create relations between abstract and concrete elements, and a layout tool that can be reconfigured to reflect the decisions made at previous stages in the design process. Designers can then specify various parameters to define how the information in the models can be used to design the user interface. This enables designers to tailor general design criteria to the specific application. It supports

the development of desktop graphical interfaces and it was one of the best tools of the second generation.

As mentioned before, the main issue underlying the third generation of model-based tools is the design of multi-device interfaces. In current practise the design of multi-platform applications is often obtained through the development of several versions of the same applications, one for each platform considered. Then, such versions can at most exchange data. This solution with no tool support is rather limited, because it implies high implementation and maintenance costs. Thus, there is a need for authoring environments able to support the development of multi-device interfaces by providing design suggestions taking into account the specific features of the devices at hand.

LiquidUI is an authoring environment whose main goal is to reduce the time to develop user interfaces for multiple devices. It is based on the user interface markup language (UIML), a declarative language that then can be transformed in Java, HTML, and WML through specific rendering software. A UIML program, with its generic vocabulary, is specific to a family of devices (such as the desktop family, the PDA family, the WAP family). There is a transformation algorithm for each family of devices. For example, using a generic vocabulary for desktop applications, the developer can write a program in UIML once and have it rendered for Java or HTML. TERESA (Mori et al., 2004) 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, including the task level, and consequently generate the concrete user interface for a specific type of platform. Currently, the tool supports user interface implementations in XHTML, XHTML mobile device, and VoiceXML (Berti and Paternò, 2003) and a version for multimodal user interfaces in X + V is under development. The tool is able to support different level of automations ranging from completely automatic solutions to highly interactive solutions where designers can tailor or even radically change the solutions proposed by the tool. The last version of the tool supports *different entry-points*, so designers can start with a high-level task models but they can also start with the abstract user interface level in cases where only a part of the related design process needs to be supported. 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 thanks to a number of automatic features that have been implemented (e.g. the possibility of links between abstract interaction objects and the corresponding tasks in the task model so that designers can immediately identify their relations). This is useful for designers to maintain 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.

An evaluation was conducted at the Motorola Italy software development centre (Chesta et al., 2003) with an early version of the TERESA tool. The experiment consisted in developing a prototype version of an e-Agenda application running on both desktop and mobile phone. Results showed similar total times for the traditional and TERESA approaches, with different distributions over the development phases and between time required by the first and the final version. The TERESA-supported method offers a good support to fast prototyping, producing a first version of the interface in a significantly

shorter time. On the other side the time required to modify it results increased. The use of the tool almost doubled required time at re-design stage, while at development stage the results show a dramatically improved prototyping performance, reducing needed time to half. This leaves a margin for further improvement, since the design time required by TERESA approach is expected to decrease as the subjects become more familiar with model-based techniques and notations. Moreover the reported slight total time increase of using TERESA with respect to using traditional approaches (on average, it was half an hour) is acceptable since it involves a trade-off with design overall quality: many subjects appreciated the benefits of a formal process supporting the individuation of the most suitable interaction techniques. For example, designers reported satisfaction about how the tool supported the realization of a coherent page layout and identification of links between pages. The evaluators noticed and appreciated the improved structure of the presentations and more consistent look of the pages resulting from the model-based approach. This is also coupled with an increased consistence between the desktop and the mobile version, pointed out by almost all the evaluators. After this evaluation a number of features in TERESA have been improved so that future evaluation should provide even better results.

### *3.2. Reverse engineering*

Various approaches can be considered when designers want to start with an existing system and obtain a conceptual description of its design in order to analyse it or derive the user interface for another platform.

Vaquita (Bouillon et al., 2002) addresses Web applications and aims at identifying presentation models for single pages, which mainly means the abstract description of the interaction techniques used in the implementation. Vaquita reverse engineers any HTML page into a presentation model expressed in XIML (Puerta and Eisenstein, 2002). The Web page is firstly cleaned in order to remove unused tags and obtain a standard XML code format. Vaquita extracts the DOM model, detects objects included in this DOM model (e.g. images, banners, controls, text), structures them into relevant categories, translates them into different levels of abstractions (Abstract Interaction Objects, Logical Windows, and Presentation Units). According to the designer's options (e.g. object inclusion or exclusion, questions, information selection), Vaquita outputs a presentation model specified in XIML. This step completes the reverse engineering part. The logical description obtained can serve as input to generate user interface code for a different target computing platform. To this end, a number of transformations can be provided depending on target platforms. These transformations are motivated by two different aims: either the reduction of the size of the original UI, as the main objective of reengineering Web pages is to render them on mobile platforms, or to replace an object by another, because the current object does not exist on the target platform. The authors of the Vaquita tool have decided to develop a new version. The new tool (ReversiXML) produces a presentation model in USIXML (Limbourg et al., 2004). In addition, the new tool should support on-the-fly reengineering. Instead of a static, off-line approach, reverse engineering of HTML pages may be done dynamically at the server level.

Another tool for reverse engineering is WebRevenge (Paganelli and Paternò, 2003). The purpose of this tool is to automatically reconstruct the task model of the Web

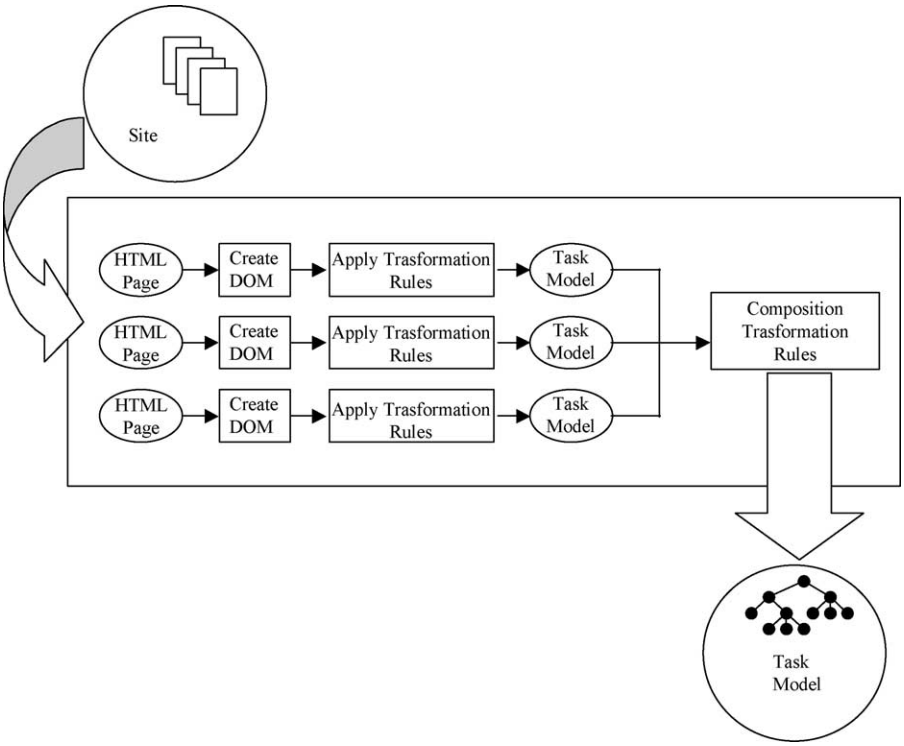


Fig. 3. The architecture of WebRevEnge.

application considered. The tool (see Fig. 3) receives as input the Web pages of the site. The site can be either in the local system or remote. The tool is able to automatically identify the pages composing it. At the beginning the user is required to indicate which page is the home page of the site to analyse. It first checks that the HTML code is well-formed and, if not, it corrects it and then creates the corresponding DOM which describes the structure of the page (all the elements contained in it). Then, following a set of rules it creates the task model associated with each page represented through the ConcurTaskTrees notation. The final part of the underlying algorithm is dedicated to identifying higher-level tasks that involve multiple pages. Thus, the resulting model is also able to describe how the current design assumes that activities should be performed even when they require interactions through multiple pages of the site. The resulting task model can be saved either in XML format or in a format that can still be subjected to modifications or adjustments by the designer using tools publicly available for this purpose.

One limitation of such tools that should be solved in the near future is the lack of support for the reverse engineering of Web sites implemented using dynamic pages.

More generally, as Fig. 4, shows various approaches can be considered when designers want to start with an existing system and derive the user interface for another platform. Transcoding mainly operates on the syntactical level and it is unable to change the design choices. A real redesign can be obtained only if the supporting environment is able

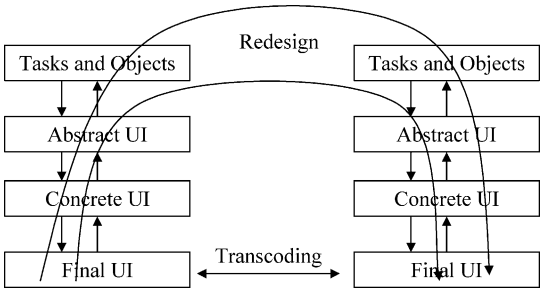


Fig. 4. Approaches to changing the design for a different platform.

to understand the logical aspects associated with the current user interface elements. The more the environment is able to go up in terms of corresponding abstraction levels, the more substantial the design choices that can be performed taking into account the characteristics of the new target device. This means that if the run-time environment is able to identify the concrete object associated with the current user interface element then it is possible to represent that specific object in a way tailored for the new device whereas if it is able to identify the corresponding abstract object then the environment can change the choice of the interaction technique to use for implementing it depending on the characteristics of the new device. However, if the environment is also able to understand the corresponding task then it can reason at this level as well and make a wider set of decisions: the task at hand can still be performed but with different modalities (different user interface elements or domain elements or even a different task structure with different subtasks associated with the same main task) or it can decide that in the new context of use the original task has no longer importance and new tasks should be enabled and supported. It is difficult to perform completely automatically this level of reasoning.

### *3.3. XML-based languages for model-based design of multi-device interfaces*

XML-based languages can be useful for representing the relevant concepts and ease their automatic manipulation. They can be structured through a set of simple and clean constructs. This type of approach supports the possibility of extending the language. Various tools are available to support manipulation of XML-based languages. In addition, they ease the exchange of information among different tools. For example, this approach allows designers to develop a model with one tool, analyse its content with another one and generate final interfaces with still another one if such tools share the support for the same XML language. Examples of projects that have addressed these issues are: *The user interface markup language (UIML)* (http://www.uiml.org/) (Abrams et al., 1994) is an XML-compliant language that allows a declarative description of a user interface in a device-independent manner. This has been developed mainly by Harmonia and Virginia Tech. However, their tools do not support the task level. How to connect task models in CTT with UIML is currently under investigation (Alì et al., 2003).

*The eXtensible Interface Markup Language (XIML)* (http://www.ximl.org/) (Puerta and Eisenstein, 2002) is a XML specification language for multiple models. This has been

developed by a forum headed by RedWhale software. Tool support is not currently publicly available. The XIML goals are: support to interface design, manipulation, organization, and evaluation; possibility of relating abstract interfaces concepts to concrete implementation objects; possibility of exploiting the information in XIML specifications through knowledge-based systems. XIML separates the user interface specification and its rendering. A single specification can be rendered in different devices. In this process aspects such as screen dimensions, presence or absence of other information, user preferences and so on are takein into account. The XIML basic document structure describes the interface in terms of its components, their relations and attributes. Each component is composed of one ore more elements. There are five fundamental types of components: tasks, domain elements, users, presentation components, dialogue components. XIML supports manipulation of concrete and abstract elements, clear separation between design and implementation, a knowledge that can be exploited at run-time, and independence of the operating system and development environment. However, the support to the manipulation of abstract elements is still quite limited. The authors of XIML explicitly state that the relations among UI models matter just as much as the models themselves, if not more. Some of the mappings between models are well-understood, for example using the domain model to select presentation elements, or using the task model to structure the dialogue model. XIML aims to leverage this understanding to provide support for user interface designers. The basic idea is to use task model, domain model, user model and style guidelines to generate automatically presentation and dialogue model. The goal of the SEESCOA project (software engineering for embedded systems using a component-oriented approach) [LC01] is to create a framework that will support run time migratable UIs that are independent of the target platform. The approach to the problem is component-based. In this component-based approach there are components that export XML descriptions that can be rendered by other components.

TERESA XML is composed of a set of XML-based languages. There is one language for the task level, one language for the abstract user interface level and one language for each platform considered at the concrete interface level. The task model is described by the CTT notation, which was the first language for task models specified by XML. The concrete interface languages share the same structure of the abstract interface language but add aspects which are specific to the platform associated. The main transformations supported in TERESA are performed in terms of the XML-based representations that are supported. From the XML specification of a CTT task model 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*. Both the XML task model and Presentation Sets specifications are the input for the transformation generating the associated abstract user interface. The specification of the abstract user interface, in terms of both its static structure (the 'presentation' part) and dynamic behaviour (the 'dialogue' part), is saved for further analyses and transformations *from abstract user interface to concrete 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. Through the last transformation the tool automatically generates the final user interface.

A recent proposal for an XML-based language for multi-device interfaces is USIXML (Limbourg et al., 2004) The language has been formally described as UML Class Diagrams. XML Schema are used instead of DTDs because they are more powerful and structured. The principle of separation of concerns is respected in USIXML because any model is self-contained and refers to other elements on demand. The task model is a full CTT model expanded with some task attributes, and with external reference to the context of use. The domain model is a full class diagram with classes, attributes, methods, and relationships (predefined and custom). It also encompasses a description of objects (i.e. instances of classes). The abstract interface level is largely expanded through Allen relationships and a systematic description in terms of actions and elements. The authors introduced a context model composed of a user model, a platform model, and an environment model. The platform model is a subset of CC/PP to preserve W3C compatibility.

## 4. New challenges for model-based tools

The increasing interest in model-based approaches stimulated by multi-device environments opens up further challanges in the coming years, in particular in the area of natural development, ambient intelligence, and multi-modal interfaces.

### *4.1. Natural development*

One fundamental challenge for the coming years is to develop environments that allow people without particular background in programming to develop their own applications. The increasing interactive capabilities of new devices have created the potential to overcome the traditional separation between end users and software developers. Over the next few years we will be moving from *easy-to-use* (which has yet to be completely achieved) to *easy-to-develop interactive software systems*. Some studies report that by 2005 there will be 55 million end-users developers, compared to 2.75 million professional developed. End user development (EUD) is a set of methods, tools, and techniques that allow people, who are non-professional developers, at some point to create or modify a software artefact. There are at least some criteria that should be supported and pursued to extend model-based approaches to the point of obtaining natural development environments: integrated support of both familiar/informal and engineered/structured specifications and effective representations supporting the analysis and highlighting of the information of interest. In fact, at the beginning of the design process many things are vague, so it is hard to develop precise specifications from scratch, especially because a clear understanding of the user requirements is a non-trivial activity. The main issue of end-user development is how to exploit personal intuition, familiar metaphors and concepts to obtain/modify a software artefact. On the one hand, natural development implies that people should be able to work through familiar and immediately understandable representations that allow them to easily express relevant concepts,

and thereby create or modify applications. On the other hand, since a software artefact needs to be precisely specified in order to be implemented, there will still be the need for environments supporting transformations from intuitive and familiar representations into precise -but more difficult to develop- descriptions. The main motivation for model-based approaches to user interface design has been to support development through the use of meaningful abstractions to avoid dealing with low-level details. Despite such potential benefits, their adoption has mainly been limited to professional designers, so there is a need for solutions that are able to extend such approaches in order to achieve natural development by enabling end-users to develop or modify interactive applications still using conceptual models, but with continuous support that facilitates their development, analysis, and use, for example with the support of vocal and sketch-based interfaces.

### *4.2. Model-based design for multi-modal interfaces*

Multi-modal interfaces are becoming more common because of various technological improvements. Most model-based approaches have only considered graphical interfaces. The few that have addressed different modalities have tackled them separately, so that the environment generates either graphical or vocal interfaces depending the designer's choice. When the synergistic use of different modalities has been considered, usually this is obtained through either ad hoc developed systems or through methods that have no tool support for actually generating the multimodal interface. We need to bridge this gap with integrated tools able to provide more general and flexible solutions.

In (Faraday and Sutcliffe, 1997) a method for designing multi-modal interfaces based on task models was presented. The method starts with the evelopment of a task model. For each task the required type of information is indicated. Then, the communication goals are associated with each task. All this information is useful for selecting the most suitable media first and then the specific user interface technique. Finally, a user validation is foreseen. The types of information can be: descriptive, spatial, operative actions, temporal, operative procedures. The communication goals are structured according to rhetorical categories: subject-informative (enable, result, cause, inform), subject-organizing (sequence, summary, condition), presentational (locate, foreground, background, emphasise). This method was applied for an application supporting ship captains to manage fire alarms. The main limitation is that no automatic tool supports it.

In model-based design of multi-modal interfaces many issues have to be addressed. The task performances can be influenced by the modality available: a set of input can require separate interactions through a graphical device, whereas such information can be provided through a single interaction by using a vocal interface. In addition, different modalities can be useful for different tasks. For example, the vocal channel is more suitable for simple or short messages, for signalling events, immediate actions, to avoid visual overloading and when users are on the move, whereas the visual channel is more useful for complex or long messages, for identifying spatial relations, when multiple actions have to be performed, in noisy environments or with stationary users.

When the goal is to obtain interfaces supporting multiple modalities (for example graphical and vocal interaction), the choice of the implementation techniques still has to consider other aspects of the platform. So, it will be different generating user interfaces

for a multi-modal desktop system or for a multi-modal PDA system because in one case the user is stationary and the graphic screen large, whereas in the other case the user can be on the move and the screen is small. Thus, the ways to provide input information or output or feedback on the input have to take into account the features of the available platform. The analysis of the space of the possible design choices and the identification of the most suitable one can be done through the CARE properties (Coutaz et al., 1995) that identify various ways to use multimodalities: complementarity (synergistic use), assignment (one specific modality is selected), redundancy (multiple modalities for the same purpose), and equivalence (there is a choice of one modality from a set of available ones).

### *4.3. Model-based support for ambient intelligence*

Models are useful not only at design time, but also at run-time. For example, they have been used to obtain more meaningful help systems (Pangoli and Paternò, 1995). Collagen (Rich and Sidner, 1998) 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. In (Einsenstein et al., 2001) there is a discussion on how model-based approaches (in particular, using a platform, a presentation, and a task model) can be used to support multi-device applications. To this end, the authors consider a hypothetical software application to create annotated maps of geographical areas. However, the discussion in the paper seems more a logical discussion on how to approach the problem, rather than the presentation of a solution which is supported by some implementation, even a prototype.

PUC (Nichols et al., 2002) is an environment that supports the downloading of logical descriptions of devices and the automatic generation of the corresponding user interfaces. The logical description is performed through templates associated with design conventions, which are typical design solutions for domain-specific applications. The approach of the Aura project to the problem is to provide an infrastructure that configures itself automatically for the mobile user, 'potentially using whichever computing capabilities are available or reachable from the current location' (de Sousa and Garlan, 2002). When a user moves to a different platform, Aura attempts to reconfigure the computing infrastructure so that the user can continue working on tasks started elsewhere. An Aura environment is supposed to run in multiple places such as home, office, car, etc. The context observer of an Aura environment detects events of interest that occur in the physical location (e.g. user is entering, user is leaving, etc.) and informs the local environment manager as well as the local task manager of these facts. The local environment manager is in charge of modelling the computing and interaction resources locally available. The task manager, called Prism, checkpoints the state of the running suppliers (i.e. services needed to support users' tasks) at a high level of abstraction. For example, for a text editing supplier, the task manager saves the file name as well as the current insertion point in the text of the source document. When the current local context observer detects that the user is leaving the Aura environment, it informs the local task manager, which checkpoints the local suppliers and causes the local environment manager to pause those services. When the user enters another Aura environment, the new local

context observer detects the fact and informs the new local task manager. In turn, the task manager reinstantiates the tasks that were suspended by finding and configuring services suppliers in the new Aura environment. So, for example, a user who was working at home using Word can carry on the task in the new environment, though possibly using a different text editor such as Emacs. In this approach, tasks are considered as a cohesive collection of applications. When a user refers to a particular task, the system automatically brings up all the applications and files associated with that task. This mechanism relieves the user from finding files and starting applications individually. Suppliers provide the abstract services that tasks are composed of. In practice, these abstract services are implemented by just wrapping existing applications and services to conform to Aura APIs. For instance, Emacs, Word and NotePad can each be wrapped to become a supplier of text editing services. Different suppliers for the same type of service will typically have different capabilities. However, the issue of how to adapt a certain interactive service to the current platform is not addressed in this research.

Another approach in this area is the Dygimes framework (Coninx et al., 2003) for developing multi-device Uis, which calls for developing a task specification enriched with the UI building blocks. This can be sufficient to generate prototype UIs useful in a user-centred design process. The time necessary to create these prototypes is short because many of the steps the designer had to do manually with traditional GUI building toolkits are now automated by the framework. For example, the transformation from the task specification to the resulting functional UIs built by the UI designer is done automatically. A micro-runtime environment offers support for rendering the created UIs independent of the chosen widget toolkit.

One important result that can be achieved through run-time transformations is user interface migration where the user interface is transferred from one device to another one at run-time, still maintaining the current state resulting from the previous user interactions. This can be very useful for users who can freely move and dynamically exploit the resources of new devices that they encounter during their movements. The migration takes into account the runtime state of the interactive application and the different features of the devices involved. Different types of runtime migration can be identified, along with different levels of complexity for each one of them:

- total, the user interface migrates entirely to a different device;
- partial, part of the user interface migrates to a different device;
- distributed, the user interface is reconfigured in such a way to support interaction through multiple devices.

The runtime migration engine exploits information regarding the interface's runtime state and higher-level information on the platform-dependent interface versions. Runtime application data are used to achieve interaction continuity, while semantic information is considered to adapt the application's appearance and behaviour to the specific device (Fig. 5).

An example of possible solutions to migration is presented in (Bandelloni and Paternò, 2004) where interfaces developed through a model-based approach are considered. The goal of this research is to support distributed migration. As it is represented in Fig. 4.

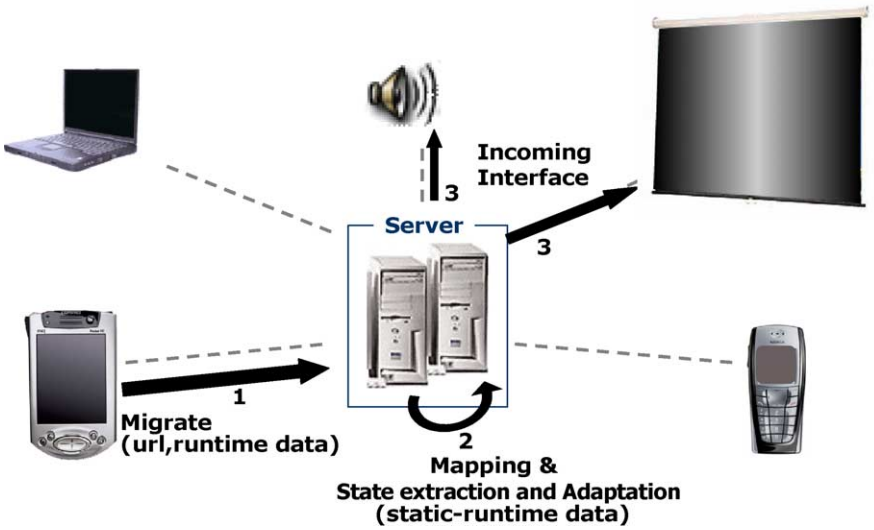


Fig. 5. Example of distributed interface.

The idea is that the user is interacting with a PDA, then he gets in a new environment with new interactive resources. The migration service is able to detect those that are more suitable to support the possible tasks and is also able to detrmine how to split the interface across such devices. So, for example it can decide to use vocal interation for some parts and a large screen for other interaction technique. To this end the state of the source interface is extracted and mapped into the two new interfaces that are activated adapting to their features.

Since the number of presentations and the tasks supported by the various platforms may be different, it is not possible to create a one-to-one correspondence between presentations for different platforms. One important issue is how to identify the presentation for the target platform corresponding to the one active on the platform requesting migration. The page to be visualized on the target device is identified using the following process: depending on the tasks to support, the tool identifies the most similar abstract presentation and then the corresponding page in the interface version for the target platform. Similarity is calculated in terms of supported tasks, the more similar the tasks supported by the two presentations (source and target) are, the more similar the presentations will be considered. Presentation similarity is the basic criterion to be considered, but under particular conditions it may not be enough. When the migrating presentation supports a task set that is associated with multiple presentations in the target version, each of which supports the same number of tasks supported by the original presentation, then a further criterion is used to decide which target presentation to activate. To this end, it is selected that supporting the task associated with the interaction object last modified, since the user is most likely to continue interaction from that point. Once the target presentation has been identified, it is necessary to calculate the state of the objects contained in the corresponding page. For this purpose, data referring to the runtime state of the interface will be associated to the corresponding tasks, then the corresponding interaction objects in

the target presentation are identified and their state is adapted with the state information. This allows obtaining, for example, that if an element in a radio-button was selected in the source presentation and the same choice is supported through a list in the target presentation then the corresponding element in the list will be automatically selected. One limitation of this approach is that currently works only on TERESA-generated interfaces.

In general, in ambient intelligence the basic point is to allow users to freely move from one environment to another and still be able to seamlessly perform their tasks with the support of a wide variety of interactive devices. This can be achieved through ubiquity, awareness, intelligence, and natural interaction. Thus, a run-time environment based on the use of task models can open up a set of more flexible and powerful solutions than those currently provided. This presumes an ability to take into account how the context of use can change in terms of interaction resources and to dynamically redesign the interface in order to preserve usability when such changes occur.

## 5. Conclusions

The most common model-based approach in software engineering, UML, provides little support to the design of the interactive components of software artefacts. UML is actually a collection of languages, including collaboration diagrams, activity diagrams, as well as use case support. These languages are intended to cover all aspects of specifying a computational system. While they have been used for the interactive part as well, they have not been expressly designed to support it and they tend to ignore some specific aspects related to the user interaction. Specific model-based approaches have been developed to support user interface designers. A proposal to integrate these two approaches is in (Paternò, 2001). As various authors have pointed out, the increasing availability of new interaction platforms has raised new interest in this approach because model-based systems can help to identify the appropriate interaction techniques without having to deal with a plethora of low-levels details. Even recent W3C standards, such as XForms (Xforms, 2004), have introduced the use of abstractions similar to those considered in the model-based community to address new heterogeneous environments.

The need for context-dependent services has raised interest in exploiting task models at run-time as well in order to deploy more usable interfaces for the enabled tasks. While in the past task models have been considered for supporting users of desktop systems, for example through context-dependent help systems, nowadays the steadily increasing availability of new types of interactive devices, in particular mobile devices, opens up new possibilities for exploiting this approach.

This paper discusses model-based approaches, with particular attention to those based on task models, tool support for their development and analysis, XML-languages supporting their use for multi-device interfaces, and the new challenges for this area.

Despite increasing interest, only a limited number of approaches to addressing such challenges have reached a sufficient degree of maturity. This is probably because of the complexity of the issues involved, which require accounting for the many aspects involved with providing user interfaces able to adapt to different devices while preserving usability. In addition, such approaches should be integrated with the rest of the architecture to allow

convenient efficient performance and scalability, especially considering the requirements of commercial service providers. However, model-based design is entering industrial practice, and the importance of usability in nomadic services calls for the design and development of intelligent infrastructures able to follow users and support them effectively. To this end, knowledge of user preferences and resource availability regarding task performance is important, as is a system that maintains some representation of user intent. This can be used to obtain intelligent environments where users can freely move about from one area to another and still be able to continue working on tasks started elsewhere, because the system has reconfigured the interface taking into account the resources available on the new devices, while at the same time maintaining its state.

## Acknowledgements

Support from the EU CAMELEON IST Project (http://giove.isti.cnr.it/cameleon.html) is gratefully acknowledged.

## References

Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J., 1994. UIml: An Appliance-independent XML User Interface Language, Proceedings of the Eight WWW Conference 1994.

Alì, F., Perez-Qinones, M., Abrams, M., 2003. in: Seffah, A., Javahery, H. (Eds.), Building Multi-Platform User Interfaces with UIML, in Multiple user Interfaces. Wiley, New York, pp. 95–118.

Annett, J., Duncan, K.D., 1967. Task analysis and training design. Occupational Psychology 41, 211–221.

Bandelloni, R., Paternò, F., 2004. Flexible Interface Migration, Proceedings ACM IUI 2004, Funchal 2004, pp. 148–157.

Barnard, P., May, J., 1995. Interactions with advanced graphical interfaces and the deployment of latent human knowledge, in: Paterno, F. (Ed.), Interactive Systems: Design, Specification, Verification. Springer, Berlin, pp. 15–48.

Beard, D., Smith, D., Denelsbeck, K., 1996. Quick and dirty GOMS: a case study of computed tomography. Human–Computer Interaction 11 (2), 157–180.

Berti, S., Paternò, F., 2003. Model-based Design of Speech Interfaces, Proceedings DSV-IS 2003. Springer, Berlin.

Biere, M., Bomsdorf, B., Szwillus, G., 1999. The Visual Task Model Bulder, Proceedings CADUI'99. Kluwer, Dordrecht.

Bodart, F., Hennerbert, A., Leheureux, J., Vanderdonckt, J., 1994. A Model-based Approach to Presentation: a Continuum from Task Analysis to Prototype, Proceedings DSV-IS'94. Springer, Berlin pp. 77–94.

Booch, G., Rumbaugh, J., Jacobson, I., 1999. Unified Modeling Language Reference Manual. Addison Wesley, Reading, MA.

Bouillon, L., Vanderdonckt, J., Souchon, N., 2002. Recovering Alternative Presentation Models of a Web Page with VAQUITA, Proceedings of CADUI'02, Valenciennes. Kluwer, Dordrecht pp. 311–322.

Coninx, K., Luyten, K., Vandervelpen, C., Van den Bergh, J., Creemers, B., 2003. Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems, Proccedings Mobile HCI 03. Springer, Berlin. LNCS N.2795, pp. 256–270.

Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., Young, R., 1995. Four Easy Pieces for Assessing the Usability of Multmodal Interaction: the CARE Properties, Proceedings INTERACT 1995 1995 pp. 115–120.

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

de Sousa, J., Garlan, D., 2002. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments, IEEE-IFIP Conference on Software Architecture, Montreal 2002.

Einsenstein, J., Vanderdonckt, J., Puerta, A., 2001. Applying Model-Based Techniques to the Development of UIs for Mobile Computers, Proceedings IUI'01: International Confrence on Intelligrent User Interfaces. ACM Press, New York.

Faraday, P., Sutcliffe, A., 1997. Designing Effective Multimedia Presentations, Proceedings ACM CHI'97 1997, pp. 272–278.

Foley, J., Sukaviriya, N., 1994. History, results, and bibliography of the user interface design environment (UIDE), an early model-based system for user interface design and development, in: Paterno, F. (Ed.), Interactive Systems: Design, Specification, Verification. Springer, Berlin, pp. 3–14.

Hartson, R., Gray, P., 1992. Temporal aspects of tasks in the user action notation. Human–Computer Interaction 7, 1–45.

Hudson, S., John, B., Knudsen, K., Byrne, M., 1999. A Tool for Creating Predictive Performance Models from User Interface Demonstrations, Proceedings UIST'99 1999, pp. 93–102.

John, B., Kieras, D., 1996. The GOMS family of analysis techniques: comparison and contrast. ACM Transactions on Computer–Human Interaction 3 (4), 320–351.

John, B., Prevas, K., Salvucci, D., Koedinger, K., 2004. Predictive Human Performance Modeling made Easy, Proceedings ACM CHI 2004. ACM Press, New York, pp. 455–462.

Kieras, D.E., 1996. Guide to GOMS Model Usability Evaluation using NGOMSL, second ed, The Handbook of Human–Computer Interaction North Holland, New York.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez-Jaquero, V., 2004. UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proceedings of the Ninth IFIP Working Conference on Engineering for Human–Computer Interaction jointly with 11th International Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004, 2004, 89–107.

Mori, G., Paternò, F., Santoro, C., 2002. CTTE: support for developing and analysing task models for interactive system design. IEEE Transactions in Software Engineering 28 (8), 797–813. IEEE Press.

Mori, G., Paternò, F., Santoro, C., 2004. Design and development of multi-device user interfaces through multiple logical descriptions. IEEE Transactions on Software Engineering 30 (8), 507–520.

Myers, B., Hudson, S., Pausch, R., 2000. Past, present, future of user interface tools. Transactions on Computer–Human Interaction, ACM 7 (1), 3–28.

Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R., Pignol, M., 2002. Generating Remote Control Interfaces for Complex Appliances, Proceedings ACM UIST'02 2002, pp. 161–170.

Paganelli, L., Paternò, F., 2003. A tool for creating design models from web site code. International Journal of Software Engineering and Knowledge Engineering 13 (2), 169–189. World Scientific Publishing.

Pangoli, S., Paternò, F., 1995. Automatic Generation of Task-oriented Help, Proceedings ACM Symposium on User Interfaces Software and Technology. ACM Press, Pittsburgh, pp. 181–187.

Paris, C., Tarby, J., 2001. A Flexible Environment for Building Task Models, Proceedings of the IHM-HCI 2001, Lille, France 2001.

Paternò, F., 1999. Model-Based Design and Evaluation of Interactive Applications. Springer, Berlin. ISBN 1-85233-155-0.

Paternò, F., 2001. Towards a UML for Interactive Systems, Engineering HCI'01, Lecture Notes Computer Science N.2254. Springer, Toronto.

Paternò, F., 2003. Models for Universal Usability, Invited Talk, Proceedings IHM 2003. ACM Press, Caen, November 2003, pp. 9–16.

Paternò, F., Leonardi, A., 1994. A Semantics-Based Approach to the Design and Implementation of Interaction Objects, Computer Graphics Forum, vol. 12, N.3. Blackwell Publisher, Oxford, pp. 195–204.

Puerta, A.R., Eisenstein, J., 1999. Towards a General Computational Framework for Model-Based Interface Development Systems, IUI99: International Conference on Intelligent User Interfaces. ACM Press, New York pp. 171–178.

Puerta, A., Eisenstein, J., 2002. XIML: a Common Representation for Interaction Data, Proceedings IUI2002: Sixth International Conference on Intelligent User Interfaces 2002.

Rich, C., Sidner, C., 1998. COLLAGEN: a collaboration manager for software interface agents. User Modelling and User-Adapted Interaction 8 (3/4), 315–350.

Szekely, P., 1996. Retrospective and Challenges for Model-based Interface Development, Second International Workshop on Computer-Aided Design of User Interfaces. Namur University Press, Namur.

Szekely, P., Luo, P., Neches, R., 1993. Beyond Interface Builders: Model-based Interface Tools, Proceedings INTERCHI'93. ACM Press.

Tam, R.C.-M., Maulsby, D., Puerta, A., 1998. U-TEL: a Tool for Eliciting User Task Models from Domain Experts, Proceedings IUI'98. ACM Press.

The CAMELEON Project, 2004. http://giove.isti.cnr.it/cameleon.html.

van Welie, M., van der Veer, G.C., Eliëns, A., 1998. An Ontology for Task World Models, Proceedings DSV-IS'98. Springer, Berlin, pp. 57–70.

Violante F., 2001. OPTA: Open Task Analysis, PhD thesis, Politecnico di Milano, Dipartimento di Elettronica e Informazione.

Wilson, S., Johnson, P., Kelly, C., Cunningham, J., Markopoulos, P., 1993. Beyond Hacking: a Model-based Approach to User Interface Design, Proceedings HCI'93. Cambridge University Press, Cambridge, pp. 40–48.

Xforms., 2004. The Next Generation of Web Forms, http://www.w3.org/MarkUp/Forms/.