SILVIA BERTI, FABIO PATERNÒ, CARMEN SANTORO

NATURAL DEVELOPMENT OF NOMADIC INTERFACES BASED ON CONCEPTUAL DESCRIPTIONS

1. ABSTRACT

Natural development aims to ease the development process of interactive software systems. This can be obtained through the use of familiar representations and intelligent environments able to map them onto corresponding implementations of interactive systems. The main motivation for model-based approaches to user interface design has been to support development through the use of meaningful abstractions in order to avoid dealing with low-level details. Despite this potential benefit, their adoption has mainly been limited to professional designers. This paper shows how they should be extended in order to achieve natural development through environments that enable end-users to develop or modify interactive applications still using conceptual models, but with continuous support that facilitates their development, analysis, and use. In particular, we discuss the application of the proposed criteria to the CTTE and TERESA environments, which support the design and development of multi-device interfaces.

2. INTRODUCTION

Two fundamental challenges for development environments in the coming years are end user development and interaction device heterogeneity. The former aims to 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 still to be completely achieved) to *easy-to-develop interactive software systems*. EUD in general means the active participation of end-users in the software development process. In this perspective, tasks that have traditionally been performed by professional software developers are transferred to the users, who need to be specifically supported in performing these tasks. New environments able to seamlessly move between using and programming (or customizing) should be designed. The other challenge is raised by the ever-increasing introduction of new types of interactive devices. Their range varies from small devices such as interactive watches to very large flat displays. The availability of such platforms has forced designers to strive to make applications run on a wide spectrum of computing devices in order to enable users to seamlessly access information and services

[Editor(s) here] (ed.), [Book Titlen here], 1—1.

regardless of the device they are using and even when the system or the environment changes dynamically. Thus, there is a need for new development environments able to provide integrated solutions to these two main challenges.

One of the goals of end-user development is to reach closeness of mapping: as Green and Petre put it (Green and Petre, 1996): "The closer the programming world is to the problem world, the easier the problem-solving ought to be.… Conventional textual languages are a long way from that goal". Even graphical languages often fail to furnish immediately understandable representations for developers. The work in Myers' group aims to obtain natural programming (Pane and Myers, 1996), meaning programming through languages that work in the way that people who do not have programming experience would expect. We intend to take a more comprehensive view of the development cycle, thus not limited only to programming, but also including requirements, designing, modifying, tailoring, etc. Natural development implies that people should be able to work through familiar and immediately understandable representations that allow them to easily express and manipulate relevant concepts, and thereby create or modify interactive software artefacts. In contrast, 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 more precise, but more difficult to develop, descriptions. In this paper, we discuss how to apply this paradigm to the CTTE and TERESA environments, which support the design and development of multi-device interfaces starting with descriptions of the logical tasks to support. Such environment is able to generate multiple interfaces adapted to the interaction resources of the various target platforms.

In the next sections we first discuss related work, then we discuss some criteria for delivering natural development environments to informal programmers. In Section 5, we provide a general introduction to the different viewpoints that can be considered when dealing with interactive systems, while in Section 6 we focus on an authoring environment, TERESA, we developed for design of multi-device interfaces. An example of application using the TERESA approach is shown in Section 7, while some conclusions have been drawn in section 8.

3. RELATED WORK

Several years ago many people argued that programming is difficult because it requires the precise use of a textual language; the hope was that environments eliminating textual languages would have been inherently easier to use. This meant to consider visual environments based on the consideration that many people think and remember things in terms of pictures. In other words, they relate to the world in an inherently graphical way and use imagery as a primary component of creative thought. On the one hand, reducing or removing entirely the necessity of translating visual ideas into textual representations can help to speed up and make easier the learning phase. Visual programming methods like conventional flow charts and

graphical programming languages have been widely studied in the last twenty years. On the other hand, end-users still have problems even when using visual environments. It seems that visual languages might be better for small tasks, but often break down for large tasks (Green and Petre, 1992). An example of adoption of the visual approach is the Unified Modelling Language (UML) (OMG, 2001), which has become the de facto standard notation for software models. UML is a family of diagrammatic languages tailored to modelling all relevant aspects of a software system; and methods and pragmatics can define how these aspects can be consistently integrated. Visual modelling languages have been identified as promising candidates for defining models of the software systems to be produced. They inherently require abstractions and should deploy concepts, metaphors, and intuitive notations that allow professional software developers, domain experts, and users to communicate ideas and concepts. This requirement is of prominent importance if models are not only to be understood, but also used and even produced by end users.

Generic languages are designed for general-purpose problems, whereas domain-specific programming languages are designed for a specific class of problems. In order to remain generic, the first type must accept a trade-off between the expressiveness of the constructs, their intuitiveness, and the need to suit a variety of needs. Languages of the second class focus on a specific type of problem. This facilitates narrowing the range of objectives and more properly defining the goals of the language. The consequence is that the language constructs are easier to learn because they are tailored to the task, while the user's background knowledge allows using high-level constructs and abstractions because some details are dealt with automatically. Such languages are able to quickly augment user productivity and some applications can be obtained through the use of a set of application-specific libraries that are able to enhance existing tools for end user development.

Natural programming has been defined as a "faithful method to represent nature or life, which works in the way the people expect" (Myers, 1998). In general, it is advisable that user interfaces are "natural" so they are easier to learn and use, and will result in fewer errors. For example, (Nielsen, 1993) recommends that user interfaces should "speak the user's language"; this means that user interfaces should include a good mapping between the user's conceptual model of the information and the computer's interface for it. For this reason, it is important to enable users to speak their language for expressing their conceptual view in a natural way. A production-oriented programming style should be more natural; a possible approach is to use a well-known real-world system as a metaphor for the computational machine in order to make the programming procedure more concrete. Many previous research systems used metaphors, such as the "turtle" in Logo. However, the use of metaphors can lead to other intrinsic problems, for example the difficulty in finding an appropriate metaphoric model for a given programming environment. In addition, metaphors do not scale well; in fact, a metaphor that works well for a simple process in a simple program will often fail to work well as that process grows in size or complexity. Then, the use of appropriate metaphors, with their capabilities

and limitations, differs widely depending on the users and their purposes. A useful method for natural programming is mixed-initiative dialogue, where the user and the system have a conversation about the program.

Another interesting direction in which end user programming can be pursued is through the use of model-based approaches, which are sometimes criticised due to problems in understanding/using them, but, on the other hand, are indisputably helpful in managing complexity, as they allow designers to focus just on the most relevant concepts. In particular, since one of the basic usability principles is "focus on the users and their tasks", it became important to consider *task* models. Task models can be useful to provide an integrated description of system functionality and user interaction. Then, the development of the corresponding artefact able to support the identified features should be obtainable through environments that are able to suggest the most effective interaction and presentation techniques on the basis of a set of guidelines or design criteria. Various solutions have been proposed for this purpose. They vary according to a number of dimensions. For example, the automation level can be different: a completely automatic solution can provide meaningful results only when the application domain is rather narrow and consequently the space of the possible solutions regarding the mapping of tasks to interaction techniques is limited. More general environments are based on the mixed initiative principle: the tool supporting the mapping provides suggestions that the designer can accept or modify. An example is the TERESA environment (Mori, Paternò and Santoro, 2004) that provides support for the design and development of multi-device interfaces, which can be accessed through different types of interaction devices. Other works have addressed similar issues but following different approaches, without considering end-user development or model-based development. One example is a completely automatic solution, which is called transcoding, in which an application written in a language for a platform is automatically transformed into an application in a language for another platform (see IBM WebSphere Transcoding) for an example of HTML-to-WML transcoding). The main problem of such approaches is that they assume that the same tasks are supported by each platform, and tend to support them in the same manner without taking into account the specific features of the platform at hand, so providing poor results in terms of usability. Aura (De Sousa and Garlan, 2002) is a project whose goal is to provide an infrastructure that configures itself automatically for the mobile user. 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. In this approach, tasks are considered as a cohesive collection of applications. Suppliers provide the abstract services, which 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. So, the different context is supported through a different application for the same goal (for example, text editing can be supported through MS Word or Emacs depending on the resources of the device at hand). TERESA provides a more flexible solution where the same application can have different interfaces depending on the interactive device available. This is obtained

by exploiting the semantic information contained in the declarative descriptions of the tasks and the interface and using transformations that incorporate design criteria platform-dependent for generating multiple interfaces (one for each potential type of platform).

4. CRITERIA FOR OBTAINING NATURAL DEVELOPMENT ENVIRONMENTS

In this section we discuss a number of criteria that can be identified to obtain natural development environments based on the use of conceptual descriptions.

4.1 Integrating informal and structured representations

The use of informal techniques is typically associated with non professional users, since it does not require any specific a priori knowledge. However, most end-user developers would benefit from a combined use of multiple representations with various levels of formality. At the beginning of the design process many things are obscure and unclear. It is hard to develop precise specifications from scratch. In addition, there is the problem of clearly understanding what the user requirements are. Thus, it can be useful to use the results of initial discussions to feed the more structured parts of the design process. In general, the main issue of end-user development is how to use personal intuition, familiar metaphors and concepts to obtain or modify a software artefact, whose features need to be precisely defined in order to obtain consistent support for the desired functionality and behaviour. In this process we should address all the available multimedia possibilities. For example, support for vocal interaction is mature for the mass market. Its support for the Web is being standardised by W3C (Abbott, 2002). The rationale for vocal interaction is that it makes practical operations quicker and more natural, and it also makes multi-modal (graphic and vocal) interactions possible. Vocal interaction can be considered both for the resulting interactive application and for supporting the development environment.

In CTTE (Mori, Paternò and Santoro, 2002), to support the initial modelling work we provide 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. To develop a task model from an informal textual description, designers first have to identify the different roles. Then, they can start to analyse the description of the scenario, trying to identify the main tasks that occur in the scenario's description and associate each task with a particular role. It is possible to specify the category of the task, in terms of performance allocation. In addition, a description of the task can be specified along with the logical objects used and handled. Reviewing the scenario description, the designer can identify the different tasks and then add them to the task list. Once designers have their list of activities to consider, they can start to create the hierarchical structure that describes the various levels of abstractions

among tasks. The hierarchical structure obtained can then be further refined through the specification of the temporal relations among tasks and the tasks' attributes and objects. The use of these features is optional: designers can start to create the model directly using our editor, but such features can facilitate the modelling work. U-Tel (Tam, Maulsby, and Puerta, 1998) provides a different type of support: through automatic analysis of scenario content, nouns are automatically associated with the objects, and verbs with the tasks. This approach provides some useful results, but it is too simple to be generalised. For example, in some cases a task is defined by a verb followed by an object.

Another useful support can be derived from the consideration that often people spontaneously use sketches and rough designs even just to improve human-to-human communication. As a matter of fact, the result of brainstorming by either one single person or a group are often 'quick and dirty' pen/paper or whiteboard –based diagrams whose main objective is to effectively and rapidly fix/communicate ideas. Such techniques are effective in that allow people to concentrate on ideas instead of being distracted by low level details or getting stuck by some rigid symbolism. In addition, the possibility of developing through sketching can be highly appreciated especially if it is possible to capture the results of such process for further analysis. Such considerations foster application areas for intelligent whiteboard systems (Landay and Myers, 2001) or augmented reality techniques, as they are capable of exploiting such natural modes of interaction/communication. In fact, such informal techniques minimise the need of using abstraction (which are mostly used with formal techniques) while specifying systems, then, especially non professional users feel particularly comfortable with them. Such systems are able to interpret sketches by recognising graphical elements and convert them into a format that can be edited and analysed by desktop tools. In addition, they should be able to recognise not only single widgets but also composite widgets or complex groups/patterns regarding structure and navigation through an analysis of spatial relations between objects. Also, the possibility of specifying reusable custom widgets and of extending the grammar through such composite widgets should be envisaged and, in order to get a real 'flavour' of the concerned interactive system, the users should be empowered also with the possibility of associating dynamic properties to such drawings. In this way, they can be enabled to intuitively outline not only the static layout of the user interface but also its dynamic behaviour. The use of such techniques paves the way for an iterative and progressive refinement process in which multiple levels of detail are progressively included by users (designers), who gradually adjust the design moving from inherently informal, imprecise and ambiguous specifications to more rigorous and detailed designs. In this way users are allowed to overcome the formal barriers that are traditionally associated with this process and smoothly bridging the cognitive gap from informal techniques to more structured methods and representations.

Furthermore, in order to make some user commands and actions more natural, it is important that user interfaces support multiple modes of interaction, for example,

enabling users to interact vocally or by means of an input device (such as key pad, keyboard, mouse, stylus etc.).

For many users it is easier and more natural to produce spoken commands than to remember the location of function in menus and dialog boxes. Moreover, by using natural language increasingly, complex commands are enabling rapid access to features that have traditionally been buried in sub-menus and dialog. For examples, the command "Apply italic, 10 point, Arial" replaces multiple menu selections and mouse clicks. Programming in natural language might seem impossible but several studies found that when non-programmers are asked to write step-by-step informal natural language procedures, many different people use the same phrases to indicate standard programming tasks (Pane & Myers, 1996). Thanks to this fact, the vocal commands recognition is less complicated and it focuses on a set of significant keywords and phrases; this technique is adopted in CTTE tool. In the CTTE tool the users can define the task model by keyboard, mouse, and vocal commands. When they decide to use vocal interaction, the system produces a welcome message that helps to understand the information to provide; then an icon (e.g. a picture of microphone) is visualised, allowing the user to turn off the input device. This feature is useful if the users may need to have a different conversation with another person or machine, need to think about their next step, or gather material or information for the next step in order to reduce errors of speech recognition. Then, while the user vocally specifies the main tasks and associated properties, the system draws the related task model, using the symbols of the ConcurTaskTrees notation (Paternò, 1999). For example, if the user says: "At the beginning the user inserts name and password in order to access the system; then the system visualises the personal page with information on areas of interest", the recognition system, by means of a specific grammar and vocabulary, analyses and interprets the vocal message. In this example the system identifies four tasks: three interaction tasks ("*Access the system*" "*Insert name*" and "*Insert password*") and one application task ("*Visualise personal*") and defining also the main properties.

Additionally, this the system defines the relationships between the identified tasks by means of some keywords like temporal adverbs, prepositions, conjunctions and so on. In our example three type of relationships are automatically recognised (show Figure 1): the first one defines the hierarchy between the task "*Access the system*", that is the parent, and "*Insert name*" and "*Insert password*", that are the children, as in the vocal command the user says the keyword "*in order to*"; the second relationship is among "*Insert name*" and "*Insert password*", and it is interleaving relationship because the task can be performed in any order without any specific constraints indeed the user says the keyword "*and*"; the last one is between "*Access the system*" and "*Visualise personal*" that is enabling relation because the first task enables the second since the user adopts the expression "*At the beginning task1, then task2*".

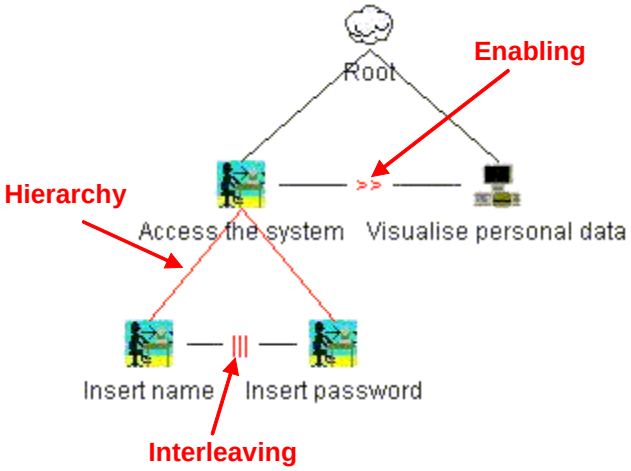


Figure 1: Example of task model generated through vocal commands.

During the conversation if the recognition system does not understand some information or has some uncertainties, it asks the user to verify the properties settings or to explain it with further details. At any time, the user is allowed to exit the vocal guided creation procedure and to operate a fully customisable development procedure.

4.2 Providing Effective Representations

Recent years have seen the widespread adoption of visual modelling techniques in the software design process. However, we are still far from visual representations that are easy to develop, analyse and modify, especially when large case studies are considered. As soon as the visual model increases in complexity, designers have to interact with many graphical symbols connected in various ways and have difficulties analysing the specification and understanding the relations among the various parts. In addition, the ever-spreading introduction and adoption of the more disparate handheld interaction devices has even more spurred designers on focussing on the recurring problem of the lack of screen space for effectively visualising large information structures on small screens still maintaining awareness of the context.

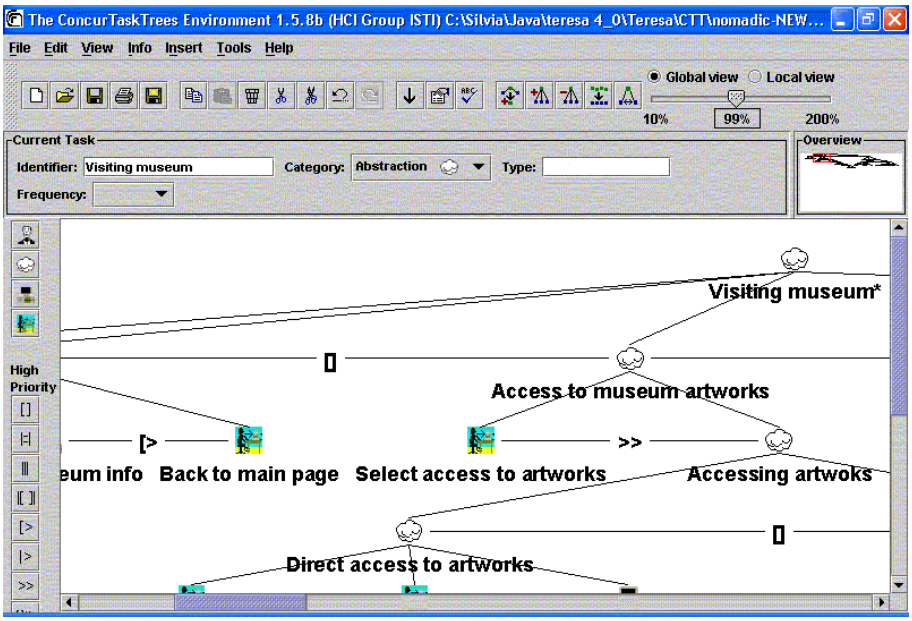


Figure 2: How CTTE provides focus+context representation.

With this regard, CTTE provides support through some representations that ease the analysis of a model. As it is shown in Figure 2, it provides a focus+context representation of the model (*focus* on details of immediate interest, plus *context* of the overall picture), allowing the user to have a detailed view of a part of the model within the main window (because only a portion can be displayed when large models are considered), still providing a small overview pane (in the right-top part) that allows the designer to maintain orientation about where the part is located with respect to the overall model within the larger window. The representation in the overview pane is limited to show the hierarchical structure disregarding details like task names and icons representing how the tasks are allocated. The correspondence between the focus window and the context window is facilitated by the hierarchical structure of the model.

In addition, the application and extension of innovative interaction techniques, including those developed in information visualization (such as semantic zooming, fisheye, two-hand interactions, magic lens…), can noticeably improve the effectiveness of the environments aiming at providing different interactive representations depending on the abstraction level of interest, or the aspects that designers want to analyse or the type of issues that they want to address.

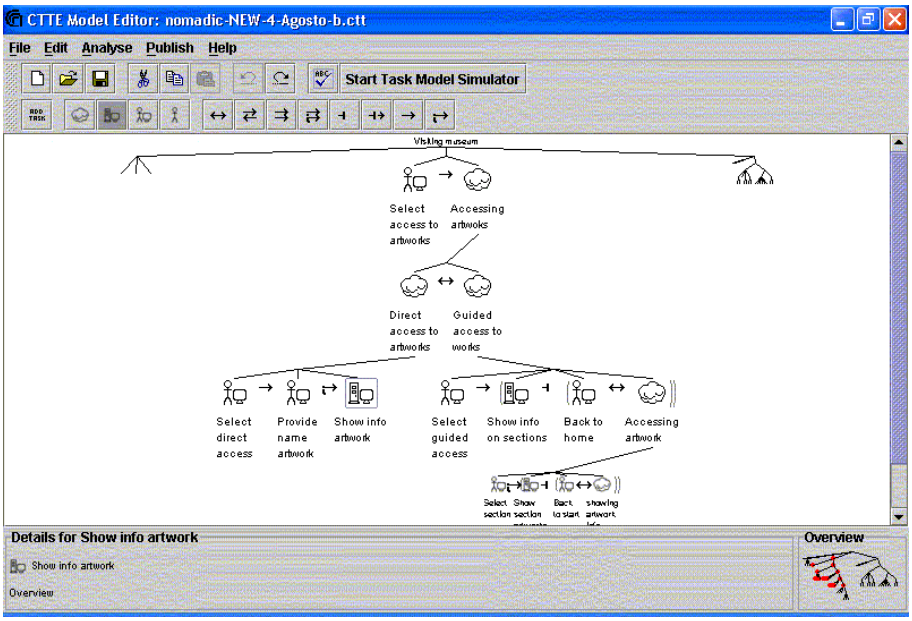


Figure 3: The fisheye-oriented representation of a task model in the new extension of CTTE.

One of the most well-known approaches in this area is the *fisheye* view (Furnas, 1981). According to this work, it is possible to calculate the Degree Of Interest (DOI) of the various elements, which is a mathematical distance function used to magnify certain areas and leaving out elements with a low DOI by means of using some de-emphasise techniques like filtering, zooming out, etc. An example of a fish-eye oriented representation of a task model is given in Figure 3, which refers to a new extension of the CTTE tool (Paternò and Zini, 2004). The advantage of this approach is that it is able to dynamically highlight the areas of interest in the model by enlarging the elements that they contain. If we compare the representations in Figure 2 with that in Figure 3 (they refer to the same task model) we can notice that in Figure 3 there is a better management of the screen space as the most interesting elements (the central ones) are more effectively highlighted. On the contrary, in Figure 2 the total amount of screen is equally divided up over all the elements, even the peripheral ones (the less interesting ones).
In addition, thanks to the automatic shrinking of the farthest elements, the new version offers a more self-contained and comprehensive view of the overall model (with respect to the old version shown in Figure 2), and the relations between the focus and the context are then maintained without necessarily visualising the whole structure.

Furthermore, an interactive system is a dynamic system, so designers need to specify how the system can evolve: the sequential constraints, the possible choices, the

interrupting activities and so on. When people analyse a specification, including the instances of the various temporal operators, they may have problems understanding the resulting dynamic behaviour. To this end, interactive simulators that show the enabled activities and how they change when any of them is performed give a better understanding of the specified behaviour. Such a feature is provided within the CTTE, in which it is possible to simulate the overall behaviour of the specification by interactively building abstract scenarios that might also help in identifying behaviours that were wrongly included in the current specification. This feature is particularly significant with large models with several concurrent operators specified, whose comprehension may result a non-trivial task especially to non professional users.

4.3 Adaptive Support for End-User Development

Interactive applications could be designed for a broad range of users and tasks and for this reason flexibility gained through adaptation has become very popular, since flexible systems are supposed to decrease cognitive workload, increase productivity and improve task efficiency. When applying this principle to automated user interface design systems some experiments have shown that adaptation could provide significant gains in performance: for example, the work in (Eisenstein and Puerta, 2000) describes an adapted, automated system for user interface design that adopts the principles of the adaptive SURF algorithm to perform interactor selection. SURF stands for: Sensitivity to a small number of examples, Understandability to the interface designer, Refinement instead of learning from scratch, and Focus on those aspects of design that are particularly suitable for automation. If, on the one hand, this has proven to be effective, it is worth pointing out that, on the other hand, especially when novice users are involved, the use of adaptation should be carefully evaluated because of the risk of adding cognitive workload to beginners who could find it difficult to use a system that continually changes.

Programming by Example (PBE; also known as Programming By Demonstration, PBD) systems exist since 1975 and a first comprehensive summary of this technique together with some computer implementations can be found in (Cypher, 1993). It is a technique for teaching the computer new behaviour by demonstrating actions on concrete examples. The goal is making automatic and speeding up the execution of repetitive tasks, via system recording of user actions and generalization of a program, so that it can be reused in analogous situations. This technique is based on the simple idea that it is easier for a beginners to learn from a demonstration of the system rather than reading the instruction procedures. This approach reveals to be effective in empowering end-users to do (just-in-time) programming. As Liebermann convincingly argues in his book (Liebermann, 2001) the power of this approach is not only the intuitiveness of the “imitation” approach, but also the capability of the system to generalise and reuse an analogous procedure in a different context.

When there are some repetitive tasks, another important possibility is to define some new commands for tailoring configuration settings for automating and accelerating such tasks. Within TERESA users have the possibility of configuring some general settings, in order to prevent the users from specifying them from scratch every time they reload a user interface description within the environment. In addition, the possibility of creating macros (i.e., a set of commands to be repeated on the same structure) by using a PBE technique is also envisaged: the environment records the user's actions of both applying a certain type of heuristics and setting a specific interactor with a specific widget, even when such settings operate at different levels of abstractions covered by TERESA. In this case the tool is supposed to be able to learn this situation and generalise from it in order to be able to reapply the same macro to a similar situation.

5. THE MANY VIEWS ON AN INTERACTIVE SYSTEM

It is important to consider the various viewpoints that are possible 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 and in the CAMELEON project (http://giove.cnuce.cnr.it/cameleon.html) a study was conducted to better structure them and understand their relations through a reference framework. 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 interaction objects supporting task performance. Only the main aspects are considered, thereby avoiding low-level details. An abstract user interface is defined in terms of presentations, identifying the set of user interface elements perceivable at the same time, and each presentation is composed of a number of interactors (Paternò and Leonardi, 1994), which are abstract interaction objects identified in terms of their semantics effects.
- *Concrete user interface*, at this point each abstract interactor is replaced with a concrete interaction object that depends on the type of platform and media available and has a number of attributes that define more concretely its appearance and behaviour.
- *Final User interface*, at this level the concrete interface is translated into an interface defined by a specific software environment (e.g. XHTML, Java, …).

To better understand such abstraction levels we can consider an example of a task: making a hotel room reservation. It can be decomposed into selecting departure and arrival dates and other subtasks concerning required services/facilities. At the abstract user interface level we need to identify the interaction objects needed to

support such tasks. For example, for specifying departure and arrival dates 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 an object showing a calendar or by pull down menus allowing a controlled selection of month, day and year. The final user interface is the result of rendering such choices also considering the specific target environment of the device: it could involve attributes like the type and size of the font, the colours available, and decoration images.

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.

6. TERESA: AN AUTHORING ENVIRONMENT FOR UBIQUITOUS INTERFACES

TERESA is a model-based environment supporting the design and development of nomadic applications (those that can be accessed through heterogeneous platforms and from different locations). The method underlying the TERESA tool is composed of a number of steps that allow designers to start with an envisioned overall task model of a nomadic application and then derive concrete and effective user interfaces for multiple devices through multiple levels of abstractions and related in-between transformations. The round trip engineering process supported by TERESA also allows maintaining links among elements at different abstraction levels, also helping users in understanding links between the various models. Further options are available within the tool, which implements the whole task/platform taxonomy (Paternò & Santoro, 2003) but provides at the same time different entry points to improve flexibility: for instance, when different devices referring to the same type of platform are considered, there is no need for a nomadic task model, since only one type of platform is involved. The tool is particularly geared towards generating Web interfaces but can be easily extended to other environments and, in order to improve interoperability with other tools, XML-based languages have been used within the tool to store user interface descriptions.

A feature particularly relevant with regard to EUD is the mixed initiative dialogue implemented within the tool, which supports different levels of automation (ranging from completely automatic solutions to highly interactive solutions where designers can tailor or even radically change the solutions proposed by the tool) suiting the various expertise levels of the target end users. In fact, when designers are experts (or the application domain is either broad or has specific aspects) then

more interactive environments are useful because they allow designers to directly make detailed design decisions. In other cases, e.g. when designers are not very skilled, the mixed-initiative interaction might provide more intelligent automatic support in order to take decisions on behalf of the user. As a matter of fact, depending on the knowledge, background and skills of the users, it is not unlikely that even basic functionality becomes quite unused for the computer-illiterate population that we have to take into account when addressing EUD issues.

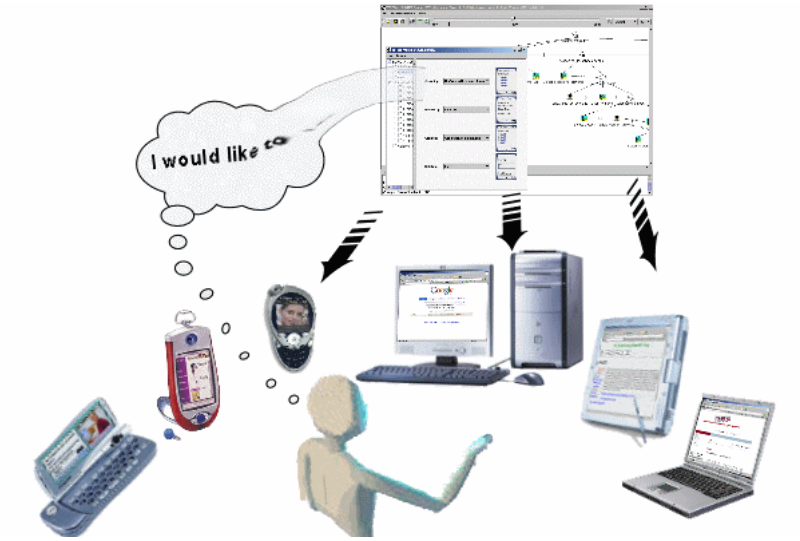


Figure 4: The approach underlying TERESA

6.1 Main Functionalities

Figure 4 shows how our approach supports natural development. Thanks to the support of input captured through various modalities (sketches, vocal input, textual scenarios) that ease the specification of the tasks to carry out, the tool first identifies the logical activities to support. Such information is then processed through a number of steps that allow designers to derive user interfaces for multiple devices:

- *High-level task modelling of a multi-context application*. In this phase designers refine a single model that addresses the possible contexts of use and the roles involved and also all the objects that have to be manipulated to perform tasks and the relations among them. Such models are specified using ConcurTaskTrees. This specification allows designers to indicate the platforms suitable for each task.
- *Developing the system task model for the different platforms considered*. Here designers have just to filter the task model according to the target platform and,

if necessary, further refine the task model, depending on the specific device considered, thus, obtaining the system task model for the platform considered.

- *From system task model to abstract user interface*. Here the tool supports a transformation 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.
- *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. In order to support generation in new user interface languages only this transformation has to be modified.

7. AN EXAMPLE OF APPLICATION

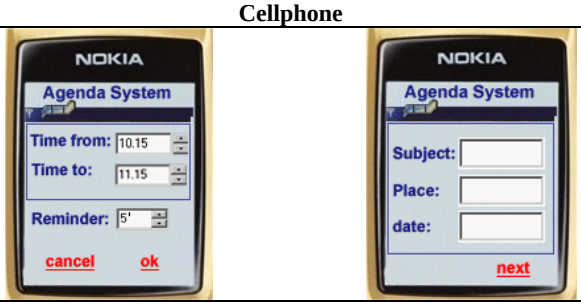
In this section we apply the approach pursued in TERESA to a digital agenda application, which presents the main features of creating and managing new appointments. As an example, we analyse the scenario of a user who wants to create a new appointment in his agenda.
First, the nomadic task model is created by the designer; the next step is to apply the filtering in order to obtain the appropriate task model for the specific platform (cellphone, desktop, PDA, vocal platform). The task model obtained can then be transformed into an abstract user interface. The TERESA tool supports various modalities for performing such a transformation, ranging from a completely automatic solution to a semiautomatic one, with various support levels. Once the abstract user interface has been obtained, the development process can evolve in different manners depending on the platform and the modality considered. Nevertheless, in all cases the tool provides designers with the possibility of changing some parameters to customise concrete interfaces.

In order to illustrate this approach, the table below shows the presentations generated to support the same task (create a new appointment) on different devices. Some differences between the various application interfaces can be highlighted: for example, on the desktop system, due to the amount of screen space available, it is possible to set all the parameters concerning a new appointment within the same presentation, while, in the mobile system, the user has to navigate through two different presentations. In the case of a VoiceXML-enabled phone, the navigation is implemented by vocal dialogs.

Moreover, further differences can be identified as far as the various supported tasks are concerned. For instance, in the desktop system it is possible to insert a description of the appointment, while this task is not available in either the mobile or voice system, due to the specific interaction characteristics of those media which both discourage the users from providing verbose descriptions that are likely to increase the total interaction time requested.

Another interesting point is represented by the different ways in which objects are arranged within the different presentations on the various platforms. At the level of abstract user interface such compositions are expressed by abstract operators which mainly refer to well known arrangement techniques (like grouping, ordering, etc.) that can be rendered, at the concrete level, with different techniques depending on the platform. For example, Table 1 shows the different implementations of the grouping operator on the various platforms: in the graphical presentations the grouped objects are marked with a distinctive symbol that permits identifying the elements in the group (see the bullets in the desktop system) or delimited within a certain region (see fieldsets in the cellphone interface), while, in the vocal interface, sounds or brief pauses are used to identify the elements' group. In addition, also the navigation between the different pages can vary depending on the number of presentations that are provided; such number might be also dependent on the amount of elements that can be rendered in a single presentation. It is not by chance that, for instance, on handheld devices the pagination process splits up into different presentations a certain amount of information that, on device with larger screen can be rendered in a single presentation, so generating a bigger number of links on every presentation. An example of this can be seen in Table 1, by comparing the navigation structure rendered on the cellphone device with respect to that displayed on the desktop system.

Desktop System

Cellphone

VoiceXML-Enabled Phone

System:"… (*grouping sound*) Now you are ready to create a new appointment. Say a subject'."
user: "meeting"
System: "Your subject is: *MEETING*. Say a place of appointment."
user: "CNR in Pisa"
System:" The place of appointment is: *CNR IN PISA*. Say the day of appointment"
user "12 December 2003"
System: "Your appointment is: *12 December 2003*. (*grouping sound*) Say the start time of appointment"
user "9 a.m."
System:" say the end time of appointment"
user "12 a.m."
System: "ok, the appointment will start at 9 and will finish at 12. *(Five seconds pause)* Say how many minutes before the appointment a reminder should be sounded.
user: "5"
System: "ok, the reminder will be sounded 5 minutes before the appointment. If you would like to insert a new appointment say *OK* , or if would like to remove the appointment say *REMOVE*….

Table 1: Examples of different interfaces generated to support the same main task.

8. CONCLUSIONS

This paper provides a discussion of how model-based design should be extended in order to obtain natural development environments. After a brief discussion of the motivations for end user development based on conceptual representations, we set forth the criteria that should be pursued in order to obtain effective natural development environments. This can be achieved by extending model-based approaches to interface development. In order to make the discussion more concrete, specific environments for model-based design (CTTE and TERESA) have been considered and we have discussed their extension for supporting natural development of multi-device interfaces. The resulting interfaces implement usability criteria taking into account the limitations of the interaction resources in the various types of devices considered.

9. ACKNOWLEDGEMENTS

We gratefully acknowledge support from the European Commission through the EUD-Net Network of Excellence (http://giove.isti.cnr.it/eud.html).

10. REFERENCES

Abbott K. R. "Voice Enabling Web Application: VoiceXML and Beyond", Apress 2002

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

Cypher, A. (Ed., 1993). Watch What I Do: Programming by Demonstration, co-edited by Halbert, D. C., Kurlander, D., Lieberman, H., Maulsby, D., Myers, B., and Alan, Turransky, 1993, *The MIT Press Cambridge, Massachusetts.*

Cooper, A. (June1995). The Myth of Metaphor.

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

Eisenstein, J. and Puerta, A. (2000). Adaptation in Automated User-Interface Design. In *Proceedings of Intelligent User Interfaces (New Orleans LA, 9-12 January 2000), ACM Press, pp. 74-81*.

Furnas, G. (1981). The FISHEYE view: A new look at structured files, technical Memorandum #81-11221-9, *Bell Laboratories, Murray Hill, New Jersey 07974, USA, 12 October 1981*.

Green, T.R.G., Petre, M. (1996). Usability analysis of visual programming environments: a 'cognitive dimensions' framework. In *J. Visual Lang. and Computing, Vol.7, N.2, pp.131-174.*

Green, T. R. G. and M. Petre (1992). When Visual Programs are Harder to Read than Textual Programs. Human-Computer Interaction: Tasks and Organisation. In *Proceedings of ECCE-6 (6th European Conference on Cognitive Ergonomics). G. C. van der Veer, M. J. Tauber, S. Bagnarola and M. Antavolits. Rome, CUD.*

Green, T. R. G., M. Petre, et al. (1991). Comprehensibility of Visual and Textual Programs: A Test of Superlativism Against the 'Match-Mismatch' Conjecture. Empirical Studies of Programming: *Fourth Workshop. J. Koenemann-Belliveau, T. G. Moher and S. P. Robertson. New Brunswick, NJ, Ablex Publishing Corporation: 121-146.*

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

Landay, J. and Myers, B. (2001). Sketching Interfaces: Toward More Human Interface Design. In *IEEE Computer, 34(3), March 2001, pp. 56-64*.

Lieberman, H., (2001). Your Wish is My Command: Programming by Example, *Morgan Kaufmann*

Lieberman, H. and Liu, H. (2003). Feasibility Studies for Programming in Natural Language. *CHI 2003 workshop on "End-user development"*.

Mori G., Paternò F., Santoro C., (2002). CTTE: Support for Developing and Analysing Task Models for Interactive System Design, *IEEE Transactions in Software Engineering, pp. 797-813, August 2002 (Vol. 28, No. 8), 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, pp.507-520, August 2004, Vol.30, N.8, IEEE Press*.

Myers, B. (1998). Natural Programming: Project Overview and Proposal. Available at *http://www-2.cs.cmu.edu/ ~NatProg/projectoverview.html*

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

Nielsen, J. (1993). Usability Engineering. *Boston, Academic Press*.

OMG (2001). Unified Modeling Language Specification, Version 1.4, September 2001; available at *http://www.omg.org/technology/documents/formal/uml.htm*

Oppermann, R. (1995). Adaptive user support: Ergonomic design of manually and automatically adaptable software. *Hillsdale, NJ: Erlbaum*.

Pane, J. and Myers, B. (1996). Usability Issues in the Design of Novice Programming Systems TR# CMU-CS-96-132. Aug, 1996. *http://www.cs.cmu.edu/~pane/cmu-cs-96-132.html*

Paternò, F. (1999) Model-based Design and Evaluation of Interactive Applications, Springer Verlag, ISBN 1-85233-155-0.

Paternò, F. (2003). From Model-based to Natural Development. In *Proceedings HCI International 2003, Universal Access in HCI, pp.592-596, Lawrence Erlbaum Associates, Publishers*.

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

Paternò, F. and Santoro, C. (2003). A Unified Method for Designing Interactive Systems Adaptable to Mobile and Stationary Platforms Interacting with Computers, *Vol.15, N.3, pp 347-364, Elsevier*.

Paternò, F. and Zini, E. (March 2004). Applying Information Visualization Techniques to Visual Representations of Task Models, ISTI-CNR *Technical Report*.

Puerta, A. and Eisenstein, J. (January 2002). XIML: A Common Representation for Interaction Data. In *Proceedings IUI2002: Sixth International Conference on Intelligent User Interfaces, ACM*.

Smith, D. C. and Cypher, A., et al. (1994). KidSim: Programming Agents Without a Programming Language. *Communications of the ACM. 37(7): 54-67.*

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