# Design and Evaluation of a Tool for Nomadic Interface Development

Cristina Chesta (1), Fabio Paternò (2), Carmen Santoro(2)

(1) Motorola Electronics S.p.a. – GSG Italy
Via Cardinal Massaia 83, 10147 Torino Italy
Cristina.Chesta@motorola.com

(2) I.S.T.I. - C.N.R.
Via G. Moruzzi 1, 56100 Pisa Italy
{ fabio.paterno, carmen.santoro}@isti.cnr.it

**Abstract.**

One of the main challenges for designers and developers of interactive software systems is to address the ever-increasing availability of new types of interaction platforms. In this paper we present a solution that brings together software engineering and human-computer interaction concepts to solve such issue. Then, we discuss such a proposal on the basis of an evaluation carried out in a software development centre for mobile applications.

## 1. Introduction

Several research activities have addressed design principles, methods and software development tools to increase systems' usability while reducing development costs. Meanwhile, a number of criteria and methodologies to evaluate their usability have been proposed. However, there is a lack of methods able to address usability issues within the software development lifecycle, in particular when nomadic applications, which can be accessed through different types of interaction platforms, are considered. Such applications need additional efforts because of the heterogeneous environments that they have to address. In fact, we assist to an ever-spreading diffusion of heterogeneous devices on the mass market, used by an increasingly variegated profile of users. As a consequence, any rigid assumption about the users and the devices is a possible root cause in creating gaps between the system developed and the actual context of use in which the interaction will take place. In addition, users empowered with and getting used to all these technologically advanced devices would like to easily access information from many types of interaction platforms.

The main goal of this paper is to discuss a method, and the associated tool, aiming at solving such issues and to report on its application in a software development centre

for mobile applications. The proposed method provides support to incorporate usability concerns throughout the development process.

The solutions currently adopted to address design and development of multi-platform applications are:

- *Use of transcoders* [I02] (for example from HTML to WML) that automatically translate the code for a type of platform to the code for another type of platform; this solution is cheap (low development cost) however usually the results are characterised by low usability because the design for a type of platform is automatically transferred to another one, which may have radically different interaction characteristics;
- *Manual development* of a version for each type of platform considered; this solution has high development and maintenance costs with no support to identify usable solutions;
- *Use of stylesheets* that according to the type of platforms allow different presentations of the user interface elements; this is useful but still far from what designers and developers need because different platforms can even support different tasks thus requiring different user interface structures.

Thus, a more general approach is needed for ensuring that the global picture at the level of the activities that should be performed interacting with the system will be conveyed to the designers, so as to allow them to take the appropriate decisions. In order to achieve this goal, and avoid the explosion of possible cases to consider, a feasible solution is identifying a number of useful abstractions highlighting the main aspects that should be considered to design effective interactive applications, which is the key point of our approach.

In this paper we first discuss the contribution that models and transformations can provide to address these new challenges. Next we discuss the approach and method (supported by our TERESA tool) that we have developed to address the development of usable solutions for heterogeneous environments. We also present how we have solved the issue of connecting the user interface software with the functional core. Then, we report on a study carried out in a software development centre for mobile applications (Motorola GSG Italy) and we discuss the lesson learnt before drawing the final remarks.

## 2. Model and Transformations for Managing the Complexity of Nomadic Applications

The most common model-based approach in software engineering (UML [BRJ99]) has paid very little attention to supporting the design of the interactive component of a software artefact. The developers of UML did not seem particularly aware of the importance of model-based approaches for designing user interfaces: UML is biased toward design and development of the internal part of software systems, and has proven its effectiveness in this. However, despite UML use cases and other notations

can effectively capture "functional" requirements or specify detailed behaviours, UML does not specifically - nor adequately - support the modelling of user interfaces aspects.

The first generation of work in this area mainly focused on how to use models to support development of desktop interactive applications, examples are Mobi-D [PE99] and Mastermind [SSC95] or how to use such models to support user interaction at run-time, still in desktop applications [RS98]. As Myers and others [MHP00] pointed out the increasing availability of new interaction platforms has raised a new interest in this approach in order to allow developers to define the input and output needs of their applications, vendors to describe the input and output capabilities of their devices, and users to specify their preferences. Then, a model-based system can choose appropriate interaction techniques taking all of these into account. Even recent W3C standards, such as XForms [W3C], have introduced the use of abstractions to address new heterogeneous environments. In particular, XForms aims to separate presentation from content through the definition of a set of platform-independent controls suitable for general-purpose use. So, to some extent it applies concepts that have long been discussed in the research area concerning model-based design of human-computer interaction. The types of abstractions supported by XForms are at the level of an abstract user interface. The task level is not explicitly addressed. Once XForms is actually supported by the major browsers we plan to have our environment generate code in this mark-up language as well.
Calvary, Coutaz and Thevenin have defined the concept of plasticity to indicate user interfaces able to adapt to different platforms while preserving usability [CCN97]: TERESA is a tool for the design and development of applications supporting such concept. The new challenges have raised the need to define XML-based languages for representing the relevant concepts and ease their automatic manipulation. Examples of projects that have addressed these issues are: The User Interface Markup Language (UIML) (http://www.uiml.org/) [APB99] 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. The eXtensible Interface Markup Language (XIML) (http://www.ximl.org/) [PE01] is an extensible XML specification language for multiple facets of multiple models in a model-based approach. This has been developed by a forum headed by RedWhale software. A simple notion of task models is supported by this approach. Tool support is not currently available. Another contribution [NMH02] has focused on a slightly different problem: remote control interfaces for appliances. Also XWeb [ONP01] introduces some kind of abstract interaction objects to support separation between functionality and interaction control elements. While these approaches have shown some interesting results, there is still a lack of general solutions able to support the various relevant abstraction levels.

Another type of approach is the use of reverse engineering techniques to obtain an abstract description of an existing interactive system for a given platform and then use it as a starting point for a new design adapted for a new platform. Examples of such reverse engineering approaches are Vaquita [BV02] and WebRevEnge [PP03]: they both start with a desktop Web site code, the former allows designers to obtain an abstract user interface whereas the latter is able to derive the correspondent task model. Both of them can be used as complementary support for our approach: once an

abstract description has been obtained our tool can help the developer to obtain a new design suitable for a different type of platform.

## 3. Tasks and multi-platform environments

In order to identify the possible design solutions when multiple platforms are considered, we have developed a taxonomy of the relations between tasks and platforms. This is based on the observation that it is neither possible nor desirable to do everything through every platform. Each platform should be associated with specific contexts of use and should be effectively used only when they occur.

More precisely, in our taxonomy we identify various cases:

- *Same task on multiple platforms in the same manner*, for example entering login and password is performed similarly in various platforms.
- *Tasks meaningful only on a single platform type*; For instance, on a mobile system users can effectively check the status of a particular flight or get road directions while driving. On the other hand, while interacting through a desktop system users wish to perform tasks such as gathering information on a company including graphical representations of geographical information, or reading a movie review while its trailer is displayed.
- *Dependencies among tasks performed on different platforms*; this occurs when the performance of a task through a platform enables or disables the performance of tasks through another platform. For example, during a city tour users can select a number of works of art. This, in turn, can enable the access to more detailed information regarding them through the desktop system.
- *Same task on multiple platforms but performed in different manner* …
    - *With different domain objects*, according to the resources available in a platform, different levels of detail can be provided regarding an argument. This means that some domain objects can be considered only if, for example, there is enough screen available.
    - *With different user interface objects*, the same task can be supported through different interaction objects according to the media and the resources available. For example, the *selection museum section* task can be performed differently: using a graphical selection through a map in a desktop system and using a list of names in a mobile phone where a little screen is available.
    - *With different task decomposition*, when a main task needs to be accomplished it can be structured differently. For example, if users want to reserve a flight seat, some systems will allow them to specify only the basic parameters (day, departure and arrival towns), while others might allow specifying a number of additional (optional) parameters, such as preferred departure and arrival time, food preferences, and so on.

- *With different temporal relations among subtasks*, in particular the type of platform considered can impose particular constraints that do not occur in a more powerful system. So, for example the small screen of a phone interface may require distributing among different sequential presentations some interaction techniques that in a desktop system can be included in a single presentation. Another example is the vocal interface that serialises interactions that can occur concurrently in a graphical interface.

## 4. The Approach

The approach we describe in this paper is supported by the TERESA (Transformation Environment for inteRactivE Systems representAtions) tool, whose design and development have been driven by a number of main requirements :

- *Mixed initiative*; the tool is able to support different levels of automation ranging from completely automatic solutions to highly interactive solutions where designers can tailor or even radically change the solutions proposed by the tool. This is important to satisfy a variety of needs: situations when the time available is short, the application domain is rather narrow, or the designer has no expertise call for completely automatic solutions. When designers are expert or the application domain is either broad or has specific aspects, then more interactive environments are useful because they allow the designer to directly make important design decisions.
- *Model-based*, as Myers and others pointed out [MHP00] the variety of platforms increasingly available can be better handled through some abstractions that allow designers to have a logical view of the activities to support.
- *XML-based*, XML-based languages have been proposed for every type of domain. In the field of interactive systems there have been a few proposals that partially capture the key aspects to be addressed.
- *Top-down*, this approach is an example of forward engineering. Various abstraction levels are considered, and we support cases when designers have to start from scratch. So, they first have to create more logical descriptions, and then move on to more concrete representations until they reach the final system. We are aware that in other cases bottom-up approaches may be preferable. For this purpose, we have also developed a tool (WebRevEnge [PP03]) that allows building task models from Web site code. It complements the approach pursued through TERESA
- *Different entry-points*, our approach aims to be comprehensive and to support the various necessities that are indicated by our task/platform taxonomy discussed in the previous section, although it may happen that only a part of it needs to be actually supported (for example, when only different brands of mobile phone are considered). In this case, there is no

need for a nomadic task model, given that only one type of platform is involved and designers can start with either the corresponding system task model or the corresponding abstract user interface.

- *Web-oriented*, the Web is everywhere, and so we decided that Web applications should be our first target. However, the approach is also valid for generating user interfaces for other types of software environments, such as Java applications, Microsoft environments, …. This simply requires extending the implementation of the last transformation (from the concrete to the final user interface) for the specific software environments.

## 5. The Method

Our method for model-based design is composed of a number of steps that allows designers to start with an envisioned overall task model of a nomadic application and then derive concrete and effective user interfaces for multiple devices (see Figure 1). Task models represent the intersection between user interface design and more formal software engineering approaches by providing designers with a means of representing and manipulating a formal abstraction of activities that should be performed to reach user goals. The main steps of our method are:

- *High-level task modelling of a nomadic application*. In this phase designers develop a single model, which addresses the possible contexts of use and the various roles involved, and also a domain model aiming to identify all the objects that have to be manipulated to perform tasks and the relations among such objects.
- *Obtaining the system task model for the different platforms considered*. In this phase the task model is filtered according to the target platform and, if necessary, further refined thus obtaining the various platform-dependent task models, which represent the input of the next step.
- *Obtaining the abstract user interface*. An abstract description of the user interface composed of a set of abstract presentations is identified through an analysis of the task temporal relationships. The presentations will be specified by means of abstract interaction objects composed through various operators (grouping, ordering, hierarchy, relation), which stand for different composition techniques (for example, the grouping operator will highlight the fact that there are objects which should be grouped together because they are closely related to each other). In order to support such transformations, we have defined an XML format for the task model language and for the abstract user interface language.
- *Deriving the concrete interface*. This phase is completely platform-dependent and has to consider the specific properties of the target platform. Every interactor is mapped into interaction techniques supported by the particular target platform, and the abstract operators also have to be appropriately implemented by highlighting their logical meaning: a typical example is the set of techniques for conveying grouping relations in desktop visual interfaces by using presentation patterns such as proximity, similarity and colour.

- *User interface code generation*, this step is completely automatic as all the design choices have already been made: the code is generated starting with the concrete interface description in the target software environment.

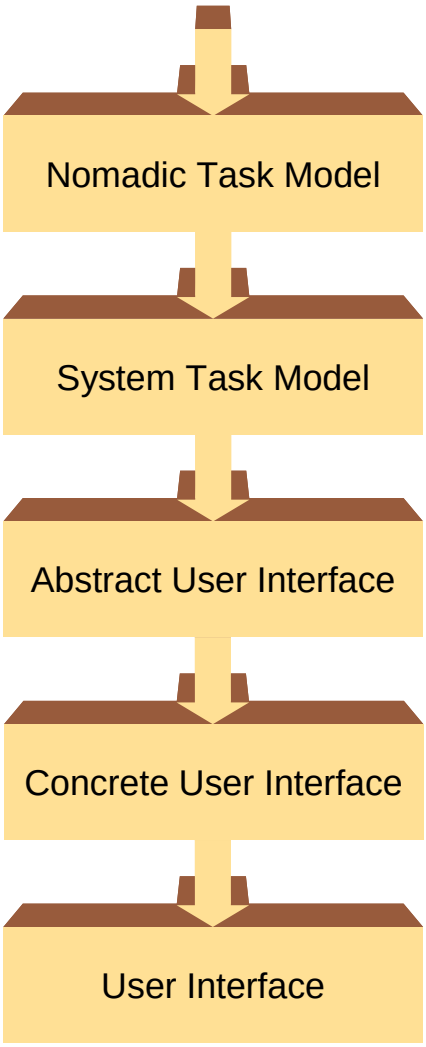


Figure 1: The Method proposed.

In the following sections we better explain such steps. In particular, we discuss how we structure the abstract user interface level and the transformation from the task model to the abstract user interface.

## 6. The Language for Abstract User Interfaces

By analysing the temporal relationships of a task model, it is possible to identify a number of Presentation Task Sets (PTS), composed of tasks that are enabled over the same period of time according to the temporal constraints indicated in the model. This is important because the interaction techniques supporting the tasks belonging to the same set are logically candidates to be part of the same presentation. In specifying the dynamic behaviour of the abstract user interface, an important role is played by abstract interaction objects associated with the *transitions*. For each presentation task set P, *transitions*(P) specifies the conditions allowing for the transition of the abstract user interface from the current presentation task set P into another presentation task set P'. The transitions can directly correspond to tasks, or, alternatively, can be expressed by means of a Boolean expression. For example, when we want to express that more than one task has to be executed in order to trigger the activation of a different presentation, an AND operator will combine the tasks.

The set of presentation sets obtained in the previous step is the initial input for building the abstract user interface specification, which is composed of interactors or Abstract Interaction Objects (AIOs) associated with the basic tasks. Such interactors are high-level interaction objects that are classified first by the type of basic task supported, then by the type and cardinality of the associated objects and lastly by presentation aspects.

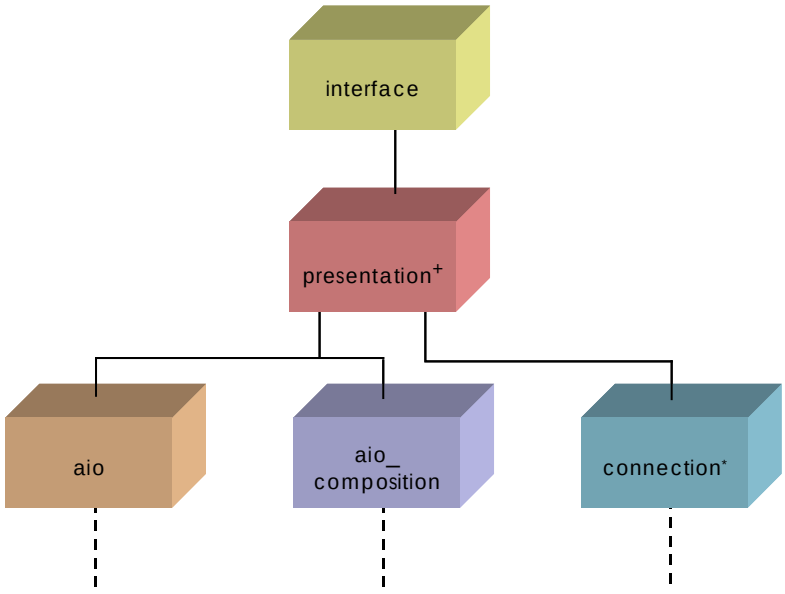


Figure 2: The structure of the TERESA abstract user interface language.

Figure 2 provides a tree-like representation of the main structure of the language we have defined for specifying the abstract user interface. An interface is composed of

one or more presentations and each presentation is characterised by a number of AIOs and their composition and 0 or more *connections* among presentations. There are two main types of objects in the abstract user interface: elementary abstract interaction objects (aio) and complex expressions (aio_composition) derived from applying the operators to these interaction objects. While the operators describe the static organisation of the user interface (in the next section we provide more detail on them), the set of *connections* describes how the user interface evolves over the time, namely its dynamic behaviour.

The abstract user interface is mainly defined by a set of interactors and the associated composition operators. The type of task supported, the type of objects manipulated and their cardinality are useful elements for identifying the interactors. In order to combine such interactors we have identified a number of composition operators for designing usable interfaces. Such operators are associated with communication goals that designers aim to achieve [MS95]:

- Grouping (G): The objective is to group together two or more elements, so this operator should be applied when the involved tasks share some characteristics. A typical situation is when the tasks have the same parent task. This is the only operator for which the position of the different operands is irrelevant.
- Ordering (O): This operator is applied when some kind of sequential order exists among elements. The most typical sequential order is the temporal one.
- Relation (R): This operator is applied when a relation exists between n elements yi, i=1,…, n and one element x. In the task model, a typical situation is when a leaf task t is at the right-hand side of a disabling operator. In this case all the tasks that could be disabled by t (at whatever task tree level) are in relation to t. This operator is not commutative.
- Hierarchy (H): This operator means that a hierarchy exists among the involved interactors. The importance level associated with the operands identifies the degree of prominence that the associated interaction objects should have in the user interface. The degree of importance can be derived from the frequency of access or from details of the application domain. Various techniques can be used to convey importance. In graphical user interfaces, one method is allotting more screen space to objects that are hierarchically more important.

## 7. Transforming tasks into abstract interaction objects

As we said before, the starting point of our method is the task model of the nomadic application that is a high-level description indicating also for each task what platforms (a platform is a set of devices that share the same interaction resources) are suitable to support it and the dependencies among tasks performed through different platforms. Then, we derive the specific task model for each platform. This is the specific starting point for the design and development of the user interface for that platform (see Figure 3).

Once we have obtained the information about tasks belonging to each presentation task set together with the transitions amongst the various sets, the next step is obtaining the description of the abstract user interface in terms of abstract user interface objects and operators described in the previous section, which requires two main steps:

- mapping the various tasks into corresponding objects of the abstract user interface;
- deriving the appropriate operators that should be applied to the various interactors.

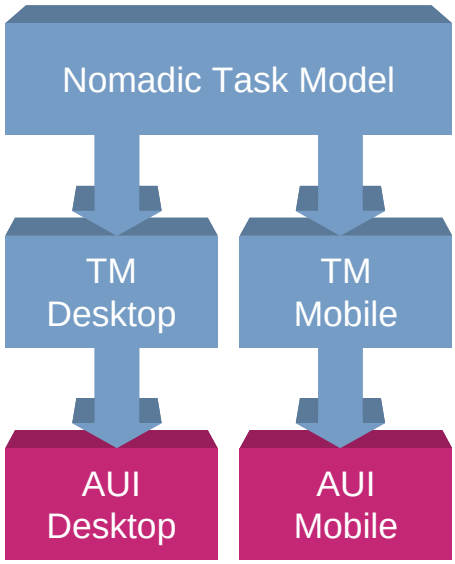


Figure 3: Example of design process for a nomadic application.

In order to map the various tasks into the related abstract user interface objects, we have to consider the information contained in the task model. First of all, the allocation of a task (whether the task is performed either through an interaction between the system and the user, or just by the application) is useful information to

identify the category of the associated abstract user interface object (respectively, interaction_aio or application_aio). Each of these interactor categories represents a class of interactors identified by the type of the task it supports. For example, an "edit" task type indicates that the corresponding interactor should allow information modification, as well as a 'selection' task type indicates that the associated interaction techniques should support the performance of this kind of activity.

Actually, these interactors are generic classes of interactors, which means that it is still possible to identify more specialised subclasses up to reaching elementary interactors. The last step for deciding the type of interactor analyses the semantics effects of the interactions to support. A number of interactors have been identified (numerical, text, description, object, position) for this purpose. Then, the combined analysis of task type and class of task objects manipulated will allow the identification of the specific, elementary interactor that is finally selected as a result of the mapping.

In addition, particular task types allow the specification of attributes that are peculiar. For selection tasks, besides specifying the type of selection supported (single or multiple), it is also possible defining the cardinality of the set of objects which the selection will be performed from (high/medium/low). For these tasks, the information about the cardinality of the set is useful to select the appropriate type of interactor able to support the selection, whereas the information about the particular type of objects manipulated is not taken into account.

The same type of rule is applied in the case of application tasks. 'Feedback' and 'Visualise' are examples of task types belonging to the application task category, they indicate the activity of presenting some results of a server-side application processing or some application data. Also in this case there are interactors suitable to support these activities.

The operators that appear in the abstract user interface are derived by analysing the ConcurTaskTrees (CTT) [P99] task model specification. In particular, not only are the CTT operators analysed, but also other attributes of the tasks (like the frequency) take part in this transformation.

In fact, on the one hand, there are some CTT operators which are directly linked with operators of the abstract user interface. It is the case when two or more tasks sequentially performed, end up in the same presentation task sets: the sequentiality at the task level can be translated, at the abstract user interface level, by applying the ordering operator.

On the other hand, other operators of the abstract user interface take into account the attributes of the involved tasks, rather than the CTT temporal operators existing amongst them. For example, it is the case of the Hierarchy operator, whose application rule strongly depends on the frequency values of the tasks involved. A high level of task frequency is indication that a task is recurrently performed, so it has greater 'importance' with respect to other tasks that are less frequently performed: the hierarchy operator is appropriate for conveying this kind of information.

### 7.1 The connection of the user interface with the functional core.

One issue is how to connect the interactive part of a software application with the functional core, the set of application functionalities independent of the media and the interaction techniques used to interact with the user. To this end, in the task model the tool automatically identifies two cases:

- the system tasks associated with functionalities related to information access in the back-end;
- when an internal functionality needs to present information to the user.

In each task specification, the objects handled to perform the tasks are indicated. Objects are classified in perceivable and application objects. The first case is identified by the lack of user interface objects in the task specification. The interactors involved in such functionality are those handling the application objects whose values should be used to send a request to the functional core. In the abstract interactor there is an attribute indicating what functionality of the core should be accessed (identified through the corresponding application task) and other attributes indicating the parameters associated to such a request. This information is further refined when moving to the concrete interface level.

In the latter case, we have application tasks that contain both perceivable and application objects. This means that the application functionality that has to present information to the user needs to communicate with the interactor handling the perceivable object associated with the task.

## 8. From the abstract user interface to its implementation

Once the elements of the abstract user interface have been identified, it is still possible to refine the description at the level of the concrete user interface. In the concrete user interface more details regarding the interactor attributes and their values are provided but it is still independent from any implementation language.

The code generation is the last step in which each interactor is mapped onto interaction techniques supported by the specific device configuration (operating system, toolkit, etc.). For example, if the object of the abstract user interface allows for a single selection from a set of objects, various implementations are available to the designer depending on the capabilities of the platform or device in question; these can include radio button menus, pull-down menus, list menus, etc.

In addition, since relationships between interactors are expressed with composition operators, they have to be appropriately implemented in order to convey their logical meaning in the final user interface. Several techniques are available for this purpose. For instance, in graphical user interfaces, a typical example is the set of techniques for conveying groupings by using classical presentation patterns such as proximity, similarity and continuity. If a different modality is used, the meaning of the same operators should be conveyed through different mechanisms. For example, in audio user interfaces, we would convey groupings with aural attributes such as pitch and volume.

As another example, a hierarchy operator for textual objects in a graphical user interface could represent important objects with larger fonts, whereas in an audio-based user interface, the hierarchy operator could represent important verbal information with a higher volume.

## 9. The Evaluation Study

In order to validate the TERESA approach and to elicit requirements, a case study related to the everyday office work has been proposed.
The envisaged e-desk application should allow people accessing office productivity applications from any place with different devices and should support:

- Accessibility from multiple platforms (PC, PDA, phone).
- Different configurations for work, travel, home, and vacation. The location could be automatically detected if the interaction platform supports positioning technologies or manually selected by the user.

The subset of services enabled will depend from both the device in use and the current location: for example, reserved docs can be viewed only if the user is in a secure location (office or home) and if he is accessing the service through a device that supports the document format.

The first version of e-desk is a minimal implementation of the requirements, including one of the planned services. The chosen service is e-agenda, a utility providing calendar and personal organizer features to record and retrieve appointments.

The choice of e-agenda was made according to following reasons. The nature of the application highlights the high strategic importance of e-desk purposes, namely the ubiquitous access to information: in a personal organizer, access to synchronized and up-to-date information from any location is a critical and valuable feature.

The application is also a fitting test-bed for nomadic interfaces topics, as calendar display offers challenging problems about interface design, emphasized by e-desk multi device issues; e-agenda will provide feedback about TERESA interface design methodologies in the form of both experimental evaluation and new requirements.

Finally, the application can take critical advantage from context-awareness. Typically, context will be used to provide relevant information and to hide non-relevant, reducing search effort; in a personal organizer, where data retrieval is the core activity as the application is in charge to remind appointments in place of the user, this can assume a critical role. E-agenda uses context to integrate this features in a mobile environment, as the reminder dynamically adapts to user location: if the user scheduled an appointment at his workplace and currently is out of office, the reminder begins one hour before the event; if otherwise the user is in office, it activates 15 minutes before the event. Context is also used to support the user in the schedule maintenance, as the application can suggest the user the best placement for an appointment according to location constraints and nearby resources. Suggestion can occur in any moment, due to changes in the detected context. Interaction with the agent follows the mixed initiative paradigm: the system helps the user to find the best

solution, in the form of a proposal not effective until explicit user acceptance; this grants the user full awareness and full control over agent actions.

The first version of e-desk is implemented as a web application with two different interfaces for desktop computers and handheld device. The application is executed on a web server with centralized storage of data; devices equipped with common web browsers and adequate screen size – like desktop PC, notebook, projector or web enabled TV – can access the desktop interface which formats application data in HTML, while smaller devices equipped with proper client – like PDA or mobile phones – may access the mobile interface, realized as XHTML Mobile Profile pages.

Interfaces are developed with the support of TERESA tool. TERESA is used to develop and maintain two consistent and adapted versions of the interface, individuating the most suitable interaction techniques for the specific client. The same set of tasks is supported with different devices. The development starts with an analysis of the interaction formalized as a CTT task model, and then TERESA processes the task model to produce the two interfaces. The output is a rough version of the final interface, which has to be refined with code snippets for dynamic data and graphical details.

Desktop interface is realized in HTML format, accessible with any common web browser. Mobile interface is realized with XHTML Mobile Profile, a recent mark-up language defined as a subset of XHTML optimized for devices with limited display capabilities; it requires a micro-browser of the last generation.

The application is realized as a Java 2 Enterprise Edition application running on a Java web server. It is structured with a 3-tier architecture, which keeps separated data storage, business logic and presentation (see Figure 4). The core application resides at business level, which is responsible for the correct management of the agenda and perform smart activities. The business layer provide an abstract, unified representation of the application, which is adapted to actual interfaces by presentation layer; interfaces never interact directly with business layer, but only through the mediation of presentation layer. Data provided by the application are integrated into the interface through JSP custom tags technology, which provides a handy format compatible with TERESA capabilities.

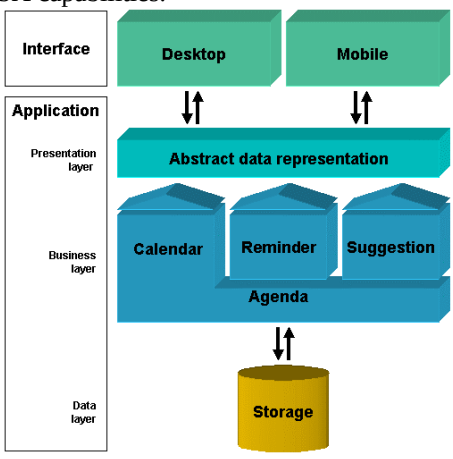


**Figure 4: e-Agenda application architecture.**

The TERESA tool has been applied to the case study described above and validated according to criteria, which take into account the double perspective of the tool itself and of the interface produced using the tool.
Starting with the ISO 9241-11 standard definition [ISO91] and Shneiderman's [S93] and Nielsen's [N94] metrics, but considering the double perspective of the tool itself versus the product realized through the tool, we identified four aspects to be evaluated and eight related requirements as listed in Table 1.

**Table 1.** Evaluation criteria

| *Aspect* | *Requirement* |
|---|---|
| Tool Interface | Intuitiveness |
| | Learnability |
| Tool Functionalities | Completeness |
| | Developer satisfaction |
| Final Product Obtained with the Tool | User Satisfaction |
| | Maintainability and Portability |
| Approach Cost/Effectiveness | Development Efficiency |
| | Integrability |

The experimental evaluation has been conducted in parallel to the tool development in order to provide a formative rather than a summative evaluation.

Two experiments have been designed in order to cover different aspects according to the criteria framework formerly exposed. Both of them refer to the common application scenario related to Business to Employee environment which has been described above.

Five subjects, selected within Motorola GSG Italy staff, were involved in the evaluation. All of them, within a range of different background and specialization, have technical knowledge and experience in software design and development, and are experienced computer users. They have been asked to participate in a 30 minutes preparation session and to dedicate 10 minutes reading the TERESA help prior to start the exercises.

The first experiment focused on tool usability and functional coverage, with the objective to highlight potential weaknesses and to provide design recommendations useful while implementing subsequent versions of the TERESA tool.

The experiment consisted in starting with a given task model created with CTTE 1.5.7 and obtaining the concrete user interface for both desktop and mobile phone using the version 1.1 of TERESA tool. The exercise goal was to realize a simple version of an e-desk application allowing three main actions: the registration to the service by inserting a username and a password, the selection of a location (workplace, home, travel or vacation), and the selection of an application from a menu. The applications offered are different in the desktop and in the mobile versions of the service. The related task tree is shown in Figure 5.

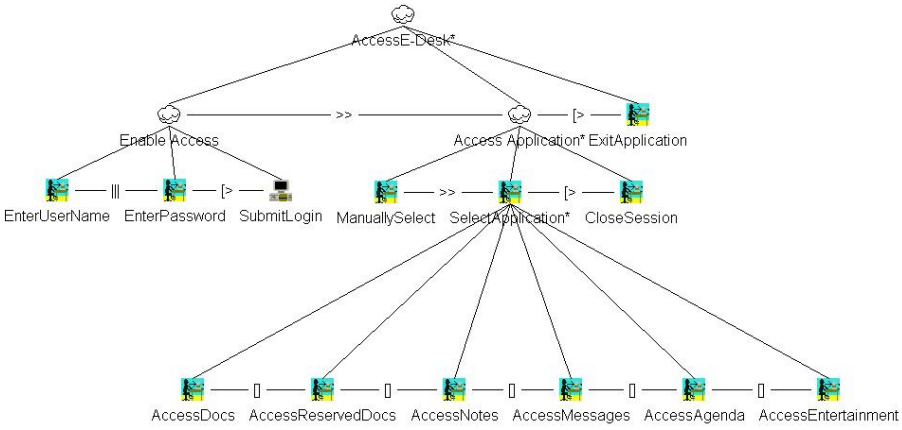


Figure 5: Task tree used in the first experiment.

The actions to be performed, such as Generate Enabled Task Sets, Generate Abstract User Interface, etc. were predefined in order to require the access to every tool menu. For each step evaluators were asked to record any difficulties they may have encountered in achieving the goal and their suggestions to improve the user interface. In addition, they were invited to provide comments about: approach, functionalities and result produced, reporting advantages/disadvantages with respect to traditional methods and providing indications on additional functionalities they would like to introduce. The first evaluation provided a number of suggestions that were considered while developing the new versions of TERESA, such as the possibility of links between abstract interaction objects and the corresponding tasks in the task model so that designers can immediately identify their relations.

A second experiment has then been conducted in order to collect more information about developer satisfaction and cost/effectiveness of the approach. The experiment consisted in developing a prototype version of an e-Agenda application running on both desktop and mobile phone and including the following functionalities: visualization of the appointments of a single day; visualization of the details of each appointment; possibility of insert/modify/delete an appointment. This had to be realized in two ways:

1. At first using traditional techniques such as a template for the design phase and Microsoft Front Page or Netscape Composer for the implementation phase
2. Then using tool-supported techniques: CTTE 1.5.7 for task tree specification and version 1.5 of TERESA tool (updated taking into account the results of the first experiment) for XHTML and XHTML mobile pages generation.

The evaluators have been required to collect quantitative metrics related to development efficiency, such as the total effort needed to complete the exercise expressed as creation or rework time and categorized by process phase, as well as the number of errors introduced. Moreover, they have been required to express their judgment on specific TERESA characteristics such as support offered to individuate the most suitable interaction techniques, support offered to compose interactors in the interface, and others aspects related to developer satisfaction and product

maintainability/portability by a rating from 1 (poor) to 5 (very good). In case of negative evaluation they were invited to provide an explanation note and suggestions for improvement.

## 10. Evaluation Results and Lessons Learnt

The evaluation resulted in an amount of data about the aspects considered. As for the first experiment the analysis has been conducted in two steps. Firstly the raw comments have been abstracted to recurrent issues aggregated by a functional criterion, counting the occurrences of each issue; this step has been conducted iteratively, in order to progressively obtain a clean taxonomy. In the second step the taxonomy obtained has been presented to the evaluators, who were requested to express for each of them a relevance assessment, either high, medium, or low; from this new data a relevance index has been synthesized for each item. The results of the analysis have been reported to the development group, which integrated them in the new version of TERESA used for the second experiment.

The new version 1.5 of TERESA has been substantially improved with respect to the first prototype. For example the effect of the heuristics used for combining together two or more PTS has been made more predictable, the AUI generation window has been redesigned in order to be intuitive and usable, the Final User Interface Generation has been improved by the introduction of preview windows.

The results of the second experiment show how developers' productivity is affected by the use of the tool. Data about time performance have been collected in each phase of the experiment and summarized through average values.

Results graphically illustrated in Figure 6, show similar total times for the traditional and TERESA approaches, with opposite impact on different phases. The use of the tool almost doubled required time at design stage, while at development stage the results show a dramatically improved prototyping performance, reducing required 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 slight total time increase is acceptable since it involves a trade-off with design overall quality: many subjects appreciated the benefits of a formal process and support to individuate 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, as well as the reduced risk to forget the formal specifications. This is also coupled with an increased consistence between desktop and mobile version, pointed out by almost all the evaluators.

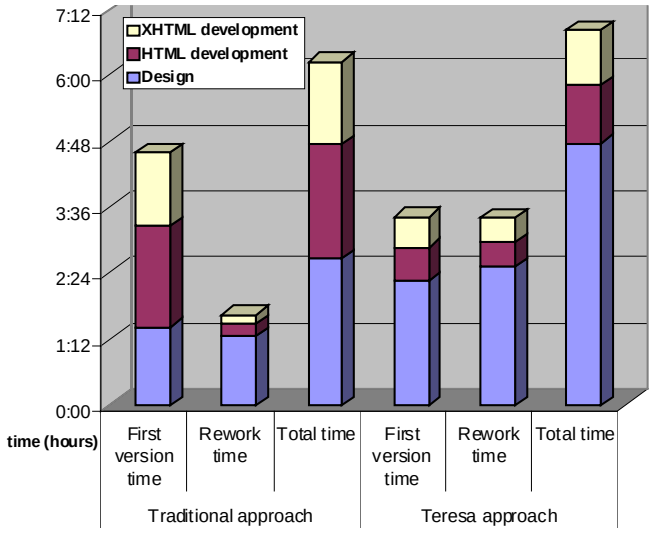


Figure 6: Comparative results on time performance.

Composition of time spent is also meaningful. The tool-supported methodology offers a very good support to fast prototyping, producing a first version of the interface in a significantly shorter time. On the other side rework time results increased. According to evaluators' comments, root cause are the high level of automation of the tool, and the current unavailability of control options, which requires the developer to successively modify the interface produced in order to reach the desired results. More in detail some subjects remarked it would be desirable having the possibility to manually redefine the connections between presentation, the presentation sets, the composition operator between interactors, the position of the interactors over the page and so on. Moreover, if changes and corrections are needed at the task model level, current prototype version of TERESA forces the user to a time-consuming iteration of the whole transformation process. The cost of corrections could be limited performing the platform filtering at a later stage of the process.

Future refinements to TERESA are then expected to consistently reduce rework time needed and to confirm the advantages of the proposed tool supported methodology.

## Conclusions and Acknowledgements

In summary, TERESA emerged from the evaluation as an appealing and promising solution for designing and developing UIs on multiple and heterogeneous devices through its use of multiple levels of abstractions. The TERESA tool leads an improvement of the user interfaces development process in at least two ways: allowing a preliminary evaluation of different alternatives and thus reducing the time

to market as well as increasing the reusability of design solutions through different devices.

At the same time the evaluation methodology and criteria we introduced appears to be general and applicable to different systems. Further activities will include additional experiments focusing on the final product obtained through TERESA and involving end users.

We gratefully acknowledge support from the European commission through the CAMELEON IST project. We also would like to thank the colleagues Cristina Barbero, Simone Martini, Bianca Russillo and Massimiliano Fliri for participating to the experimental evaluation and for the useful discussions.

## References

[APB99] Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., Shuster, J. *UIML: An Appliance-Independent XML User Interface Language*, Proceedings of the 8th WWW conference, 1999. Available at http://www.harmonia.com/resources/papers/www8_0599/index.htm

[BP03] Berti S., Paternò F., Model-based Design of Speech Interfaces, Proceedings DSV-IS 2003, Madeira, June 2003, Springer Verlag.

[BV02] Bouillon, L., Vanderdonckt, J., Retargeting Web Pages to other Computing Platforms, Proceedings of IEEE 9th Working Conference on Reverse Engineering WCRE'2002 (Richmond, 29 October-1 November 2002), IEEE Computer Society Press, Los Alamitos, 2002, pp. 339-348.

[BRJ99] Booch, G., Rumbaugh, J., Jacobson, I., *Unified Modeling Language Reference Manual*, Addison Wesley, 1999

[CCN97] Calvary, G., Coutaz, J., Thevenin, D., A Unifying Reference Framework for the Development of Plastic User Interfaces, Proceedings Engineering Human-Computer Interaction, pp.173-192, 2001.

[EVP01] Einsenstein, J., Vanderdonckt, J., Puerta, A. *Applying Model-Based Techniques to the Development of UIs for Mobile Computers*, Proceedings IUI'01: International Conference on Intelligent User Interfaces, pp 69-76, ACM Press, 2001.

[I02] IBM WebSphere Transcoding Publisher, http://www.ibm.com/software/webservers/transcoding/

[ISO91] ISO9241-11 Ergonomic requirement for office works with VDT's – guidance on usability. Technical report, International Standard Organisation, 1991.

[MHP00] Myers, B., Hudson, S., Pausch, R. *Past, Present, Future of User Interface Tools*. Transactions on Computer-Human Interaction, ACM, 7(1), March 2000, pp. 3-28.

[MPS03] Mori, G., Paternò, F., Santoro, C., "Tool Support for Designing Nomadic Applications", Proceedings ACM IUI'03, Miami, ACM Press.

[MPS02] Mori, G., Paternò, F., Santoro, C., *CTTE: Support for Developing and Analysing Task Models for Interactive System Design*, IEEE Transactions on Software Engineering, pp. 797-813, August 2002 (Vol. 28, No. 8).

[MS95] Mullet, K., Sano, D., *Designing Visual Interfaces*. Prentice Hall, 1995.

[N94] Nielsen, J. *Usability Engineering*, Morgan Kauffman, San Francisco, 1994.

[NMH02] Nichols J., Myers B., Higgins M., Hughes J., Harris T., Rosenfeld R., Pignol M., "Generating Remote Control Interfaces for Complex Appliances," *Proceedings of UIST'2002*, Ottobre 2002, pp. 161-170. Available at http://www-2.cs.cmu.edu/~jeffreyn/papers/pucUIST2002.pdf

[ONP01] Olsen D., Nielsen S.T., Parslow D., "Join and Capture: A Model for Nomadic Interaction" *Proceedings 14th annual ACM symposium on User interface software and technology UIST'01*,ACM, 2001, pp. 131-140. Available at http://icie.cs.byu.edu/ICE/LabPapers/JoinCapture.pdf

[P99] Paternò, F., Model-Based *Design and Evaluation of Interactive Application*. Springer Verlag, ISBN 1-85233-155-0, 1999.

[PL94] Paternò, F., Leonardi, A. *A Semantics-based Approach to the Design and Implementation of Interaction Objects*, Computer Graphics Forum, Blackwell Publisher, Vol.13, N.3, pp.195-204, 1994.

[PP03] Paganelli, L., Paternò, F., A Tool for Creating Design Models from Web Site Code, International Journal of Software Engineering and Knowledge Engineering (to appear).

[PS02] Paternò, F., Santoro, C., *One Model, Many Interfaces*, Proceedings Fourth International Conference on Computer-Aided Design of User Interfaces, pp. 143-154, Kluwer Academics Publishers, Valenciennes, May 2002.

[PE99] Puerta, A., Eisenstein, J., *Towards a General Computational Framework for Model-based Interface Development Systems*, *Proceedings ACM IUI'99*, pp.171-178.

[PE01] Puerta, A., Eisenstein, *XIML: A Common Representation for Interaction Data*, *Proceedings ACM IUI'01*, pp.214-215.

[RS98] Rich, C., Sidner C., *COLLAGEN: A collaboration manager for software interface agents*, User Modelling and User-Apadted Interaction, 1998, 8(3/4), pp.315-350.

[SSC95] Szekely, P., Sukaviria, P., Castells, O., Muthukumarasamy, J., Salcher, E., *Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach*. In *Engineering for Human-Computer Interactio*n, L.J. Bass and C. Unger (eds), Chapman & Hall, London, 1995, pp 120-150.

[S93] Schneiderman, S. Designing the User Interface. Strategies for Effective Human-Computer Interaction. Addisson-Wessley, Reading (MA), third edition, 1998.

[W3C] XForms – The Next Generation of Web Forms, http://www.w3.org/MarkUp/Forms/