A Process-Oriented Interactive Design Environment for Automatic User-Interface Adaptation

Margherita Antona
Anthony Savidis
Institute of Computer Science
Foundation for Research and Technology—Hellas (FORTH)

Constantine Stephanidis
Institute of Computer Science
Foundation for Research and Technology—Hellas (FORTH) and
Department of Computer Science, University of Crete, Greece

In the design of user interfaces that adapt automatically to end-users and usage contexts, designers should be prepared to cope with large design spaces; these spaces will be necessary to accommodate design constraints posed by the diversity in the target user population and the various contexts of use. Adaptation-oriented design is considered a powerful design recipe addressing the compelling requirements of customization, accessibility, and high quality of interaction. Under this perspective, it must be carefully planned, designed, accommodated, and conducted within the life cycle of interactive systems, from the early exploratory phases of design and prototyping to evaluation, implementation, and deployment. Despite recent progress, the practice of designing automatic user-interface adaptations remains difficult, due to intrinsic complexity of the task and the current limited expertise of designers and practitioners. Toward overcoming such a difficulty, this article presents MENTOR, a tool providing (a) practical integrated support for all phases of adaptation design, through appropriate editing facilities; (b) practical support for a “smooth transition” from design to development through the availability of automated verification mechanisms for the designed adaptation logic, as well as the automated generation of “ready-to-implement” interface specifications; and (c) support for the progressive accumulation of design cases and of the related design experience and knowledge, in particular regarding adaptation.
1. INTRODUCTION

The increased importance of user-interface design methodologies, techniques, and tools in the context of the development and evolution of the Information Society has been widely recognized in recent years in the light of the profound impact that interactive technologies are progressively acquiring on individuals’ lives and activities and of the difficulty in developing usable and attractive interactive products and services (e.g., Winograd, 2001; Winograd, Bennett, De Young, & Hartfield, 1996). As the Information Society develops further, the issue of user-interface design becomes even more prominent when considering the notions of universal access (Stephanidis, 2001b) and universal usability (Shneiderman, 2000), aiming at the provision of access to anyone, from anywhere and at anytime, through a variety of computing platforms and devices, to diverse products and services.

Design for Universal Access in the Information Society has been defined in recent research efforts as design for diversity (Stephanidis, 2001b), and methods, techniques, and codes of practice have been proposed that enable to proactively take into account and appropriately address diversity of users, contexts of use, and technological platforms in the design of interactive artifacts. One of the main concepts proposed in such a context is that of intelligent user-interface adaptation (Stephanidis, 2001b). In a Universal Access perspective, adaptation needs to be “designed into” the system rather than decided upon and implemented a posteriori, implying that the outcome of the design process is not a “singular” design but a design space populated with appropriate alternatives, together with the rationale underlying each alternative, that is, the specific user- and usage-context characteristics for which each alternative has been designed.

Despite the progress that has been made, however, the practice of designing for diversity remains difficult due to intrinsic complexity of the task and the current limited expertise of designers and practitioners in designing interfaces capable of automatic adaptation. Therefore, there is an emerging need to develop support tools for appropriately structuring the design process and to help designers in producing and documenting interface alternatives and adaptations.

This article discusses design support for user interfaces that exhibit intelligent adaptation to diverse user-, context-, and platform-related characteristics and proposes a tool-based approach to the provision of process-oriented support for user-interface design that facilitates the design of user- and context-dependent adaptations; this has been instantiated in an interactive design environment called MENTOR. MENTOR embodies a design method that has been developed in recent years with the explicit purpose of supporting the design of universally accessible interfaces that exhibit intelligent adaptation behavior, namely the Unified User Interface Design Method (see section 3). MENTOR’s objectives are

- The provision of practical integrated support for all phases of Unified User Interface Design by appropriately guiding the process and structuring the outcomes of creative design steps through appropriate editing facilities.
- The provision of practical support for a smooth transition from design to development of Unified User Interfaces through the availability of automated verifi-
cation mechanisms for the designed adaptation logic, as well as the automated generation of ready-to-implement interface specifications, including the adaptation logic.

- The provision of support for the progressive accumulation of Unified User Interface Design cases and of the related design experience and knowledge, particularly regarding adaptation, by offering means for extending and reusing (parts of) past design cases.

This article is organized as follows. Section 2 discusses current trends in tool-based support for user-interface design and provides a rationale for the design and development of MENTOR. Section 3 briefly describes the Unified User Interface Design Method, and section 4 describes the MENTOR functionality and interactive features. Subsequently, section 5 presents a design case study performed using MENTOR, and section 6 discusses the user evaluation of MENTOR as well as its validation in example design cases. Section 7 concludes the article with a discussion of the lesson learned and future work.

2. CURRENT TRENDS IN HUMAN–COMPUTER INTERACTION DESIGN SUPPORT

This section attempts to provide an overview of current, relevant HCI design support methods and tools, focusing in particular on the degree to which the latter are suitable for supporting the design of automatic user-interface adaptations in a Universal Access perspective. Toward this purpose, model-based design tools environments and approaches to design rationale encoding are considered.

2.1. Model-Based Design Environments

A variety of user-interface design tools and environments are based on modeling relevant user-interface-related knowledge. Examples of model-based design tools are ADEPT (Markopoulos, Wilson, Pycock, & Johnson, 1992; Wilson, Johnson, Kelly, Cunningham, & Markopoulos, 1993), Tamot (Lu, Paris, & Vander Linden, 2002; Paris, Vander Linden, & Lu, 2001), CTTE (Mori, Paternò, & Santoro, 2002), Teallach (Griffiths et al., 1999), UIDE (Foley, Kim, Kovacevic, & Murray, 1991; Sukaviriya, Foley, & Griffith, 1993), HUMANOID (Luo, Szekely, & Neches, 1993; Szekely, Luo, & Neches, 1992), MECANO (Puerta, 1996), MASTERMIND (Browne, Davila, Rugaber, & Stirewalt, 1997; Szekely, Sukaviriya, Castells, Muthukumarasamy, & Salcher, 1995), MOBI-D (Puerta & Eisenstein, 1998), TRIDENT (Bodart et al., 1997), TADEUS (Stary, 2000), VTMB (Biere, Bomsdorf, & Szwillus, 1999), and Euterpe (van Welie, van der Veer, & Eliëns, 1998). Model-based design environments vary according to a variety of factors, including overall objectives, number and types of models, provided design support, and design outcomes (Pinheiro da Silva, 2000; Szekely, 1996). Almost all design environments include graphical editors for task hierarchies and other models. Representation languages
may highly vary in their expressive power and in the degree to which their semantics are precisely formalized. In the majority of model-based systems, the user does not need to directly edit the underlying representation language. However, some systems provide textual editable specifications of some models.

Design environments also differ with respect to the number and type of design activities that are automated (e.g., automatic generation of the models, automatic generation of help and documentation, and automatic generation of the implemented user interface). In some tools, the entire interface is automatically produced (Puerta & Eisenstein, 1999). Overall, design automation appears to be a difficult task due to the large dimensions of the design space and the difficulty in automatically selecting appropriate designs. To overcome this problem, automatic generators are usually bound to generate specific or limited types of interfaces. Many environments provide tools for checking models at various levels, based on formal reasoning methods (e.g., syntactic correctness of the task model, temporal properties of models, incompleteness or ambiguity of the specification, model consistency, etc.).

More recently, the emergence of multiplatform user interfaces (Seffah & Javahery, 2003) has lead to deeper investigation of computing platforms as well as environmental and contextual factors influencing the use of interactive systems (e.g., Dey & Abowd, 2000). Various model-based approaches for supporting the design of multiplatform user interfaces have been proposed (e.g., Calvary, Coutaz, & Thevenin, 2001; Eisenstein, Vanderdonckt, & Puerta, 2001; Paternò & Santoro, 2002). These approaches usually address issues related to handling the constraints posed by different platforms and screen sizes, selecting and generating appropriate presentation structures, and managing contexts of use.

Despite recognized benefits concerning model-based approaches, the impact of many of the aforementioned tools on design practices has been limited so far (Eisenstein & Puerta, 2000; Pinheiro da Silva, 2000; Puerta, 1996; Szekely et al., 1995). Many tools have been criticized for lack of usability due to their complexity and limited integration in the overall development life cycle.

### 2.2. Approaches to Design Rationale Encoding

*Design rationale* has been defined as a representation of the reasoning behind the design of an interactive artifact (Buckingham Shum, 1996), documenting design decisions, as well as their justification, other alternatives considered, and argumentation leading to each decision (Lee, 1997).

Argumentation-based design rationale is claimed to have the potential to play several useful roles in design, such as encouraging deliberation and explicit consideration of alternatives, structuring and analyzing novel design problems, keeping track of decisions, communicating design reasoning within projects, maintaining consistency in decision making, tracking progress in projects and identifying recurring and unresolved issues, transferring design knowledge between projects with similar rationales, supporting the building of cumulative design knowledge bases, and assisting with the integration of perspectives from multiple stakeholders on
decisions (Buckingham Shum, 1996). Semiformal representations are often used to represent argumentation. The most commonly used representation is Questions, Options, and Criteria (Bellotti, 1993; MacLean, Young, Bellotti, & Moran, 1991). Systems supporting computer-based encoding and processing of design rationale are reported in Ball, Lambell, Ormerod, Slavin, and Mariani (1999); Burge and Brown (2000); Conklin and Burgess-Yakemovic (1995); and Fischer, Lemke, McCall, and Morch (1995).

Despite the claimed benefits, design rationale encoding is poorly practiced by user-interface designers, due to obvious factors such as time pressure but also usefulness and usability issues (Karsenty, 1996). In addition, designers are not likely to devote time to recording design rationale if this process is not felt as an integral part of design and adequately supported (e.g., Conklin & Burgess-Yakemovic, 1995; Fischer et al., 1995). Notably, most model-based design approaches are not concerned with capturing design rationale issues. However, such a requirement is emerging as a desideratum for more effective design support tools and environments (Puerta, 1999).

2.3. Requirements for Adaptation Design Support in Universal Access

In the context of universal access, model-based approaches are considered, in principle, a promising category of interface tools for intelligent adaptation, because they potentially incorporate computable user and design models (Savidis & Stephanidis, 2004b). In general, however, the capabilities for adaptation design of existing tools are limited because, on one hand, the ability for designing and implementing new presentation artifacts is largely reduced due to flexibility problems whereas, on the other hand, available presentation models are often restricted to graphical user interfaces. Another limit of current model-based approaches with respect to designing intelligent adaptation is related to their artifact orientedness and lack of instruments for capturing and managing users and context diversity and for supporting documented reasoning on diversity. Although some environments embed user or context models, these models are usually targeted toward capturing general and typical characteristics and are not equipped for representing individual differences or possible changes over time.

As design in a Universal Access perspective is an extremely complex activity, different types of support are likely to be required in different phases of design, with no single category of design support tools preferable to others. In addition, in such a context, design support tools cannot substitute for creative design work, as their capability to automate parts of the design activity is constrained by the inherent complexity of design activities. In the context of Universal Access, the concept of design rationale acquires a meaning different from the one prevalent in existing methods. Traditionally, design rationale mainly represents argumentation about design alternatives and assessments before reaching final design decisions. In the context of Universal Access, the objective is to capture the design rationale associated to alternative artifacts (Savidis & Stephanidis, 2004a) by recording the different user attributes and design objectives underpinning the final design decisions.
Table 1 summarizes the main divergences between currently prevalent approaches to HCI design support and the requirements for supporting adaptation design that emerge from the previous discussion. The design and development of MENTOR, which embodies the Unified User Interface Design Method (see section 3), have been based on these aforementioned requirements (see section 4).

### 3. UNIFIED USER INTERFACE DESIGN

The Unified User Interface development methodology (Savidis & Stephanidis, 2004a, 2004b; Stephanidis, 2001a) has been proposed as a complete technological solution toward Universal Access based on intelligent user- and usage-context interface adaptation. In this context, a Unified User Interface exhibits the following software engineering properties:

1. It embeds representation schemes for user- and usage-context profiles while it accesses individual user- and usage-context profiles (e.g., repositories, servers, files, etc.) during runtime.
2. It encompasses alternative implemented dialogue components, even for the same subtasks, appropriately associated to distinct values of user- and usage-context parameters. The need for such alternative dialogue components is identified during the design process, when, given a particular design context, for differing user- and usage-context attribute values, alternative design artifacts are deemed as necessary to accomplish optimal interaction.
3. It embeds decision-making capability to select the most appropriate dialogue components according to the particular user- and usage-context profiles during runtime while it supports interaction monitoring and inferences for profile modifications during interaction.

Overall, a unified interface is dynamically assembled from appropriately selected constituent interface components, relying on user- and usage-context-oriented decision making, thus demonstrating (a) user-adapted behavior through user awareness and (b) usage-context-adapted behavior through context awareness.

### Table 1: Current Approaches to HCI Design Support Versus Requirements for Supporting Adaptation Design

<table>
<thead>
<tr>
<th>Design process</th>
<th>Current HCI Design Support Approaches</th>
<th>Requirements for Adaptation Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling</td>
<td>Trends towards design automation</td>
<td>Support for creativity</td>
</tr>
<tr>
<td>Design domain</td>
<td>Domain, interface and dialogue modeling</td>
<td>Modeling of adaptation factors and adaptation logic</td>
</tr>
<tr>
<td>Design rationale</td>
<td>Usually GUIs</td>
<td>Open-ended</td>
</tr>
<tr>
<td>Formality</td>
<td>Design argumentation</td>
<td>Adaptation rationale</td>
</tr>
<tr>
<td>Automated support</td>
<td>UI representation languages</td>
<td>Adaptation logic</td>
</tr>
<tr>
<td></td>
<td>Design critique and dialogue simulation</td>
<td>Verification of adaptation logic</td>
</tr>
</tbody>
</table>

*Note.* HCI = human–computer interaction; GUI = graphical user interface; UI = user interface.
A particularly important aspect of developing unified user interfaces concerns the design. A design method, called the Unified User Interface Design Method (Savidis & Stephanidis, 2004b), has been elaborated as a part of the overall methodology to provide a process-oriented recipe for the design of Unified User Interfaces. The Unified Interface Design Method is a hybrid process-oriented design method enabling the organization of diversity-based design decisions around a single hierarchical structure, purposefully developed to support the design of Unified User Interfaces. It proposes a specific design process to address the management of an evolving design space, in which alternative design artifacts can be associated to variations of design problem parameters.

In this context, the process of designing Unified User Interfaces leads not to a singular design outcome but to a structured design space populating systematically multiple alternative designs. It collects and appropriately represents alternative designs, along with conditions under which each design should be instantiated at runtime (i.e., an adaptation-oriented design rationale). The Unified User Interface Design Method encompasses a variety of techniques such as task analysis, abstract design, design polymorphism, and design rationale.

The next subsections discuss representational issues in Unified User Interface Design and provide an overview of the process underlying user-interface design and the related outcomes.

### 3.1. The Space of User- and Context-Related Design Parameters

An essential prerequisite for conducting Unified User Interface Design is the conceptual categorization of diversity aspects in all relevant dimensions (users, context of use, access terminal/platform), and the identification of the target design parameters for adaptation). There is no predefined or fixed set of attribute categories or values. For example, attributes and related values may vary according to the target user groups considered in a design case or the involved contexts of use or interaction platforms. Therefore, attributes and values are chosen as part of the design process, and designers are free to use case-specific design parameters, to experiment with different sets of them, and to create their own (partial) taxonomies according to the type of design cases they address and the target user population. Accumulated experience in designing for diversity is expected to lead progressively to the identification of more commonly valid classifications of design parameters. For example, human abilities relevant for determining alternative choices in a design case are likely to affect similar design cases. Some examples of attribute classes that designers may consider in Unified User Interface Design are general computer-use expertise, domain-specific knowledge, role in an organizational context, motor abilities, sensory abilities, and mental abilities. Examples of potential context attributes are acoustic noise and light sources, whereas examples of potential relevant platform attributes are processor speed, memory, secondary storage, peripheral equipment, resolution, screen physical size, and graphics capabilities.
Combinations of design parameters in the form of triads <User profile, Platform, Context> constitute execution contexts. Appropriate design alternatives should be designed so as to facilitate specific task execution context(s).

3.2. Polymorphic Task Hierarchies

The basic representation adopted in Unified User Interface Design is called polymorphic task hierarchy and combines (a) hierarchical task analysis, (b) design polymorphism (i.e., the possibility of assigning alternative decompositions to the same [sub]task if required based on [combinations of] design parameters), and (c) user-task oriented operators. Figure 1 depicts an example of polymorphic task hierarchy, illustrating how two alternative dialogue styles for a “Delete File” task may be designed.

Alternative decomposition “styles” are depicted in the upper part of the figure, and an exemplary polymorphic decomposition, which includes physical design annotation, appears in the lower part. The hierarchical decomposition adopts the original properties of hierarchical task analysis, enriched with the capability to differentiate and represent design alternatives for the same task, mapping to varying design parameters through polymorphism. Task operators are based on the power-

FIGURE 1 The polymorphic task hierarchy concept (adapted from Savidis & Stephanidis, 2004a).
ful Communicating Sequential Processes language for describing the behavior of reactive systems (Hoare, 1978) and enable the expression of dialogue control flow formulae for task accomplishment.

In a polymorphic task hierarchy, the root represents design abstractions, whereas leaf nodes represent concrete interaction components. Polymorphic decomposition leads from abstract design pattern to a concrete artifact. Three categories of design artifacts may be subject to polymorphism on the basis of user- and usage-context parameter values:

- User tasks, relating to what the user has to do; user tasks are the center of the polymorphic task decomposition process.
- System tasks, representing what the system has to do, or how it responds to particular user actions (e.g., feedback); in the polymorphic task decomposition process, they are treated in the same manner as user tasks.
- Physical designs, which concern the interface components on which user actions are to be performed; physical interface structure may also be subject to polymorphism.

User tasks, and in certain cases system tasks, are not necessarily related to physical interaction but may represent abstraction on either user or system actions. System tasks and user tasks may be freely combined within task “formulas,” defining how sequences of user-initiated actions and system-driven actions interrelate. The physical design, providing the interaction context, is associated with a particular user task and provides the physical dialogue pattern associated to a task-structure definition. Hence, it plays the role of annotating the task hierarchy with physical design information.

Each alternative polymorphic decomposition is called a decomposition style, or simply a style. Alternative task subhierarchies are attached to their respective styles. Polymorphism constitutes a technique for potentially increasing the number of alternative interface instances represented by a typical hierarchical task model. There is, therefore, a fundamental link between adaptation capability and polymorphism.

3.3. The Design Process

The Unified User Interface Design process realizes a hierarchical decomposition of tasks, starting from the abstract level by incrementally specializing in a polymorphic fashion toward the physical level of interaction. In this process, different designs are likely to be associated with different (combinations of) user- and usage-context attribute values.

It starts from abstract- or physical-task design, depending on whether or not top-level user tasks can be defined as being abstract. An abstract task can be decomposed either in a polymorphic fashion, if user- and usage-context attribute values require different dialogue patterns, or following a unimorphic decomposition scheme. In the latter case, the transition is realized via a decomposition action,
leading to the task hierarchy decomposition state. Polymorphic decomposition, on the other hand, leads to the design alternative subhierarchies state.

Reaching this state means that the required alternative dialogue styles have been identified, each initiating a distinct subhierarchy decomposition process. Hence, each such subhierarchy initiates its own instance of polymorphic task decomposition process. While initiating each distinct process, the designer may start from either the abstract task design state or the physical task design state. The former is pursued if the top-level task of the particular subhierarchy is an abstract one, whereas the second occurs in the case that the top-level task involves physical interaction.

From this state, the subtasks identified need to be further decomposed. For each subtask at the abstract level, there is a subtask transition to the abstract task design state. Otherwise, if the subtask involves physical interaction means, a subtask transition is performed to the physical task design state.

Physical tasks may be further decomposed either in a unimorphic fashion or in a polymorphic fashion. These two alternative design possibilities are indicated by the “decomposition” and “polymorphism” transitions, respectively.

In summary, these are the rules to be applied in polymorphic task decomposition:

- If a given task does not involve physical interaction, start from abstract task design and:
  - apply polymorphism, if decision parameters impose the need for alternative styles on user/system tasks and/or physical structure
  - apply decomposition, when alternative designs are needed for the same style
- If a given task involves physical interaction, start from physical task design and
  - apply polymorphism, if decision parameters impose the need for alternative styles on user/system tasks and/or physical structure
  - apply decomposition, when an alternative design is needed to realize the same style

In Unified User Interface Design, one of the key issues is to decide in which cases the diversity in design parameters leads to different concrete interface artifacts. Designers should take care that every decomposition step satisfies all constraints imposed by the combination of target user- and usage-context attribute values. Polymorphic decomposition is required when different styles are appropriate for different execution contexts, based on the designer’s decision. Differentiation decisions can be based on consolidated design knowledge, if available (e.g., design guidelines for specific target user groups, target platforms, etc.), on the results of surveys or evaluation experiments, and so on. There is no automatism in deciding when and how adaptation should be applied in the final interface. This important aspect of the design process leads to providing the designed Unified User Interfaces with an adaptation logic suitable for runtime execution.
3.4. Adaptation-Oriented Design Rationale

In the context of Unified User Interface Design, the notion of design rationale has a fundamentally different objective with respect to well-known design space analysis methods. As already mentioned, previous approaches to design rationale mainly represent argumentation about design alternatives and assessments before reaching final design decisions, whereas in the case of Unified User Interface Design, the rationale records the different user- and usage-context attributes motivating the final design decisions. This obviously does not exclude recording for future reference the underlying decision criteria (e.g., guidelines followed, performed experiments, etc.).

In the depicted process, the following primary decisions need to be taken:

- At which points of a task the hierarchy polymorphism should be applied, based on the considered (combinations of) user- and usage-context attributes.
- How different styles behave at runtime; this is performed by assigning to pair(s) of style (groups) design relationships.

These decisions need to be documented in a design rationale recorded by capturing, for each subtask in a polymorphic hierarchy, the underlying design logic, which directly associates user-/usage-context parameter values with the designed artifacts.

Such a rationale should document

- Related task
- Design targets leading to the introduction of the style
- Supported execution context
- Style properties
- Design relationships with competing styles

In Figure 2, an instance of such documentation record is depicted, adopting a tabular notation.

Styles can be evaluated and compared with respect to any design parameter (e.g., performance measures, heuristics, user satisfaction, etc.). Evaluation or comparison results can form part of the design rationale as annotations.

Four fundamental relationships among alternative styles (concerning the same polymorphic artifact) have been identified, reflecting the way in which artifacts may be employed during interaction for an individual user in a particular context (see Figure 3). These relationships in fact express the adaptation runtime behavior of the designed interface, reflecting real-world design scenarios, and are motivated by the observation that different styles are not always mutually exclusive, even if they correspond to different (combinations of) design parameters values, as there are cases in which it is meaningful to make artifacts belonging to alternative styles concurrently available in a single adapted interface instance.
3.5. Design Outcomes

Summarizing, the outcomes of the Unified User Interface Design Method include (a) the polymorphic task hierarchy; (b) the design space populated by the produced physical designs; and (c) for each polymorphic artifact in the task hierarchy, a design rationale recording its runtime adaptation logic based on user- and context-related parameters.

The Unified User Interface Design Method does not require any specific format or notation for delivering the aforementioned outcomes, and designers are free of using any convenient encoding on a case basis or according to their personal preferences. For example, polymorphic task hierarchies can be visualized through graphlike structures annotated with text, and design rationales can be encoded into tables. Physical design can also be conducted and delivered using any suitable de-
The Unified User Interface Design Method, briefly described in the previous section, is recognized to require a higher initial effort and investment than traditional HCI design approaches (Savidis & Stephanidis, 2001c), as current design practices do not usually involve the identification of relevant design parameters, the design of alternative interface instances, the rationalization of a complex design space, and the final delivery of a complete interface adaptation logic. Tool support, including facilities for design reuse, is considered particularly important in this respect. This section describes the design and implementation of MENTOR, a support tool for the process of Unified User Interface Design, which has been developed to address the following objectives:
- Provision of an appropriate instrument for widening and improving the practice of adaptation-based user-interface design (from learning to practice). In this respect, the tool should target the community of interface designers and not assume deep knowledge of the Unified User Interface Design Method or particular HCI modeling techniques and support designers more experienced in adaptation design in effectively performing their work.

- Provision of practical integrated support for all phases of Unified User Interface Design, by appropriately guiding the process and structuring the outcomes of creative design steps through appropriate editing facilities for (a) encoding declarations (signatures) of design parameters attributes and related value spaces, (b) creating profiles of adaptation conditions (i.e., macros of instantiated combinations of design parameters), (c) encoding polymorphic task hierarchies, and (d) attaching information to artifacts (nodes) in such hierarchies.

- Provision of practical support for a “smooth transition” from design to development of Unified User Interfaces through availability of automated verification mechanisms for the designed adaptation logic as well as the automated generation of “ready-to-implement” interface specifications, including the adaptation logic.

- Provision of support for progressive accumulation of Unified User Interface Design cases and of related design experience and knowledge, in particular regarding adaptation, by offering means for extending and reusing (parts of) past design cases.

Figure 4 depicts the overall interactive environment of MENTOR, which is composed of four main editing facilities—the Design Parameters Editor, the Profiles Editor, the Polymorphic Task Hierarchy Editor, and the Properties Editor—corresponding to the main phases of the Unified User Interface Design process. Addressing the requirements reported in Table 1, the functionality of the tool reflects
the different aspects and phases of the method’s conduct and provides alternative ways for facilitating the method’s practice by less expert and more expert designers. MENTOR provides contextual support for the creative process and easy proposal–critique–modification loops while allowing different design proposal encoding strategies.

MENTOR poses no constraints on the interfaces that can be designed by using it or on the types of adaptation that can be designed. The tool does not force automation of creative parts, which are identified in the definition of relevant design parameters, the conduct of polymorphic task decomposition, and the elaboration of an adaptation rationale. As the steps involved in Unified User Interface Design are iteratively strictly interrelated, creativity support implies the possibility of following, as alternatives or in combination, different strategies for proceeding in incremental and interconnected design choices. The design of MENTOR, therefore, reflects the need of using these facilities in an iterative and interrelated fashion and of providing a “contextual” view and composition of a design case. On the other hand, MENTOR automates the less creative and more formal aspect of the method and (a) the verification of adaptation logic and (b) the automated generation of design specifications.

Only the adaptation logic is formally specified in MENTOR, satisfying the requirement of reducing formality as much as possible. All other parts of design cases, including the polymorphic task hierarchy, are encoded informally. The formal and informal parts are seamlessly integrated into the interaction environment. A very simple expression language (a fragment of DMSL) is used for specifying adaptation conditions.

The following subsections describe the fundamental phases of Unified User Interface Design, along with the related support provided by MENTOR.

### 4.1. The Design Parameter Editor

The Design Parameters Editor of MENTOR supports the encoding of flat (nonhierarchical) declarations (signature) of design parameter, encoded in attribute-value pairs following the specification of the DMSL language. These constitute the “vocabulary” for defining the “adaptation space” of the unified user interface under design. Therefore, the Design Parameters Editor has been designed in such a way as to allow easy editing of parameters while preserving signature consistency. According to the syntax of the DMSL, three types of design parameters values are allowed: (a) enumerated—values belong to a list of (more than two) strings specified by the designer; (b) Boolean (i.e., values \textit{true} or \textit{false}); and (c) integer, which is specified by supplying minimum and maximum bounds of an integer range allowed as a value. Value ranges define the space of legal values for a given attribute.

Figure 5 depicts the Design Parameters Editor, which displays parameter attributes, type, and value ranges into a three-column table. Parameters can belong to two different domains, namely the user domain, referring to parameters representing user characteristics, and the context domain, referring to parameters representing characteristics of the context of use and interactive platform(s) of the unified
user interface under design. The parameters appearing in Figure 5 refer to a hypothetical design case for a Delete File task, taking into account sighted and blind users, users with various degrees of motor ability in upper limbs, and novice and experienced users. The attributes and values that are defined therefore represent such a task context and are limited to the relevant user characteristics. In a different design case, different attributes or different value types could be used at the designer discretion. For example, in a design case addressing people with different types of vision impairments, the attribute Vision could be of type enumerated and a set of values could be defined according to the type of visual impairments considered (e.g., blind, color-blind, partially sighted, etc.). The Design Parameters Editor therefore has the role of supporting designers in defining specific design parameters according to the specifications of each design case.

Designers can add any number of new design parameters and, subsequently, edit the type and range of values. To preserve consistency, it is not possible to assign the same attribute to different parameters, although the same values can be assigned to different parameters. Deletion of a design parameter is possible if such parameter is not used in any profile or style condition. The tool also offers the possibility of importing design parameters from other design cases to support reusability of previous designs (see Figure 6).

Consistency checking between parameters already declared in the current design case and parameters selected for importing is automatically performed by the system according to the following policy:

- If the attribute name of a parameter selected for importing is not identical to the attribute name of any of the parameters already declared in the current design case, then the parameter is imported.
- If a parameter has an attribute name identical to the name of a parameters already declared in the current design case but a nonidentical type or values range, then the user is requested to confirm the overwriting of the currently defined parameter.
If a parameter has identical name, type, and value range with any of the already existing parameters in the current design case, it is not imported.

4.2. Profile Editor

One of the fundamental aspects of Unified User Interface Design, as already mentioned, is the support for associating alternative designs to different combinations of user and context design parameters and producing an adaptation logic determining the adaptation behavior of the interface at runtime. To support such a process, MENTOR provides an Editor for creating profiles (or macros) of users and context instances in the form of combinations of design parameters, thus facilitating the preparation of adaptation conditions ready to use in polymorphic task decomposition (see sections 2.2 and 2.3). Semantically, profiles are partial descriptions of users or contexts in the space defined by current design parameters and refer to sets of users or contexts for which they hold true at runtime.

Each profile is therefore constituted by a name (e.g., blind user) and an associated condition (e.g., User.vision == False). Conditions are formulated in a very simple expression language, specifically the condition fragment of DMSL for adaptation decision making in Unified User Interfaces.

Atomic conditions in DMSL have the form <Attribute> <Operator> <Value>, where

- **<Attribute>** is constituted by a prefix specifying the parameter domain (i.e., User. for user-related parameters, and Context. for context-related parameters) followed by any of the design parameter attributes defined in the current design case (e.g., User.vision)
- **<Operator>** represents an equality (==), inequality (!=), or comparison (>), >=, <, <=) relation between an attribute and its assigned value. Only equality and inequality operators can be used with enumerated and Boolean attributes. According
to the DMSL syntax, the equality operator is used for simplicity reasons for both value assignment in the case of enumerated and Boolean attributes and for logical comparison in the case of integer attributes.

- `<Value>` corresponds to an instance of the values declared as appropriate for a specific attribute. With enumerated attributes, `<Value>` indicates one of the allowed enumerated values as declared in the Design Parameters Editor (e.g., `User.Expertise == Expert`). With Boolean attributes, `<Value>` is either True or False (e.g., `User.Vision == True`). Finally, with integer attributes, `<Value>` can be a constant integer within the bounds specified in the Design Parameters Editor for the specific attribute (e.g., `User.Age >= 18`), or, alternatively, an arithmetic expression, which may contain variables referring to the value of another attribute (either instantiated in the same profile or not; e.g., `User.Age == 2003 – User.YearOfBirth`).

Atomic conditions can be negated by proposing the negation operator `NOT` (e.g., `NOT User.Expertise == Expert`). Finally, atomic conditions can be combined into complex conditions through the `AND` and `OR` operators (e.g., `User.Vision == True AND User.Expertise == Expert, User.Age > 18 OR User.Expertise == Expert`). Ordinary parentheses may be used to establish operators’ precedence.

Figure 7 depicts the standard appearance of the Profile Editor. Apart from directly creating and editing new profiles, alternative ways of introducing profiles are supported. Similarly to design parameters, profiles can be imported into the current design case from a pre-existing one. The consistency policy for profile importing is as follows:

- A profile can be imported in the current project only if the attributes included in its associated condition are declared in the current project and the value assigned to each attribute falls into the range of legal attributes for that attribute in the current project. For example, a profile with a condition `User.Vision == True` can not be imported into a design case where either the attribute `Vision` is not defined or the type of such an attribute is not Boolean.

![Stereotypes](image)  
**Figure 7** The standard appearance of the Profile Editor.
A ‘legal’ profile whose name and condition are not identical to the name and condition of any of the profiles included in the current project is directly imported.

- If the name or condition of a “legal” profile is identical to the name/condition of one of the profiles already defined, then the user is requested to confirm the overwriting of the existing profile with the imported one.

In addition, two or more profiles can be merged into a new profile, the condition of which is automatically composed by the conditions of the merged profiles through the AND or the OR operator (see Figure 8). For example, the profiles Expert User, with the condition User.Expertise == Expert, and Sighted User, with the condition User.Vision == True, are AND-merged into the profiles Sighted Expert User with the condition User.Expertise == Expert AND User.Vision == True.

Easy and syntactically correct editing of profile conditions is supported through the provision of a Condition Editor, which has been designed to facilitate beginner users of MENTOR to acquire familiarity with the DMSL expression language. The Condition Editor, when activated, practically guides designers to composing syntactically correct conditions by contextually and progressively activating interactive elements on the interface according to the condition composition phases (atomic vs. complex, as previously defined in this section). In addition, it completely automates text input, through the use of combo boxes and buttons, so as to avoid the use of undefined attributes or values as well as typing errors (see Figure 9).

4.3. The Hierarchy Editor

The Hierarchy Editor allows designers to perform polymorphic task decomposition and encode results in a hierarchy. Figure 10 depicts an overview of the Editor.

The Editor’s main function is to guide the decomposition process by contextualizing the available decomposition actions according to the category of
design artifact selected in the hierarchy. Figure 11 presents an example of such contextualization for unimorphic and polymorphic tasks, respectively. In the Hierarchy Editor, artifacts can also be deleted, copied, cut, and pasted. When an artifact is deleted, its children are also deleted to preserve the correctness of the resulting hierarchy (as the children of the deleted task may not be appropriate children of the deleted task father’s node). Artifacts can be pasted only as children of nodes that admit the category of the copied/cut artifact as decomposition. For example, it is not possible to paste a style as a child of a unimorphic artifact. Pasting results in producing a copy of the copied/cut artifact at the selected point in the hierarchy, including all its properties (see next section), with the exception of properties that are related to the position of the artifact in the hierarchy, which are updated according to the point in which the artifact is pasted.
4.4. Properties of Design Artifacts

Part of the Polymorphic Task decomposition process consists in assigning specific properties to artifacts in the polymorphic hierarchy. Different categories of artifacts involve different properties, some of which are discussed later in this section, and are particularly important for the purposes of the overall adaptation design of the resulting interface. In MENTOR it was considered particularly important for artifact properties to be editable “in context” while also working on the polymorphic task hierarchy. Therefore, the Properties Editor appears as a floating window in the overall tool environment, thus ensuring that the Polymorphic Task Hierarchy Editor is not hidden while the user is editing properties and always displays the data related to the currently selected artifact in the Polymorphic Task Hierarchy Editor. Properties are shown in a simple tablelike form, which can be directly edited in place.

Figure 12 depicts the appearance of the Properties Editor when a unimorphic task artifact is selected in the Polymorphic Task Hierarchy Editor. The three properties are specified textually in place. Changing the artifact’s name in the Properties Editor also updates the name in all places where it appears in the Polymorphic Task Hierarchy Editor. Temporal relations of subtasks are also edited in textual form, because the types of operators and temporal relations notation can vary from case to case (see section 3.3).

Figure 13 depicts the appearance of the Properties Editor when a (unimorphic) physical design artifact is selected in the Polymorphic Task Hierarchy and the
A minieditor for attaching physical design images is activated. Images will need to be separately created (e.g., drawings produced in drawing environments, scanned paper designs, snapshots of prototypes, etc.). Any image format can be used. It is also possible to attach more than one image. Full-size view of attached images is also possible in a separate viewer (see Figure 14).

The most important piece of information to be attached to styles concerns the user and context parameter instantiations that define the style appropriateness at runtime (i.e., the user- and context-related conditions under which the style is to be displayed during adaptation). Style conditions are therefore at the heart of Unified User Interface Design. In MENTOR, they are defined as properties of styles using the same expression language used for the definition of profiles, which, in fact, constitute preprepared style conditions (see section 3.2). The Property Editor allows three different ways of defining style conditions:

- By free text typing.
- By selecting (multiple) profile(s), previously defined using the Profile Editor (see the left side of Figure 15), which can be further edited and modified if necessary.
By using the MENTOR Condition Editor (see the right side of Figure 15), which is displayed in the minieditor. The Condition Editor is the same as the one used in the Profile Editor (see section 4.2).

In MENTOR, adaptation relations among styles are formulated as properties of polymorphic artifacts. Figure 16 depicts the appearance of the Properties Editor when a polymorphic artifact is selected in the Polymorphic Task Hierarchy. Binary adaptation relations are constructed by sequentially selecting:

- The first relation argument (i.e., one of the styles defined as children of the currently selected polymorphic artifact).
- The type of the relation (choosing among the four available types in the Unified User Interface Design Method, i.e., incompatibility, compatibility, augmentation, and substitution).
The second relation argument (another style among the ones defined as children of the currently selected polymorphic artifact).

4.5. Verification Facilities

MENTOR includes facilities for verification of the correctness of the adaptations embedded into the designed unified user interface by providing means to facilitate designers in “debugging” their adaptation design and to make sure that the delivered adaptation logic can reliably be embedded directly into the unified user interface development cycle. Toward this end, it is necessary to identify possible “adaptation failures” in the designed interface that relate to problems in the adaptation logic. The adopted approach is based on the hierarchical properties of Polymorphic Task decomposition and on the logical properties of the language used for adaptations formulation (i.e., the DMSL language and in particular its condition fragment) and on the consideration of possible mismatches in the design-implementation transition. Table 2 summarizes such verification facilities along with the addressed mismatches. Such facilities include

- Lexical and syntactic checker on DMSL condition expressions and automatic generation of DMSL rules.
- Formal verification of DMSL expressions satisfiability.
- Formal verification of hierarchical consistency and subsumption of DMSL condition expressions in a polymorphic task hierarchy.
- Formal verification of properties of style conditions according to the type of selected design relation.

The lexical and syntactic checker and the satisfiability verification of DMSL conditions are available in both the Profile Editor and Properties Editor (when a style is selected in the Polymorphic Task Hierarchy), where DMSL conditions may be composed.
The verification of hierarchical relations among styles in the polymorphic task hierarchy is also available in the Properties Editor. A style in the task hierarchy in fact (implicitly) inherits the conditions on its ancestor styles. Each polymorphic decomposition applies within the design context (i.e., specified conditions) of a higher level polymorphic decomposition in the same hierarchy branch, if present. The inherited condition is automatically filled in by the Property Editor in the condition cell of the related style. For the resulting adaptation logic to be correct, the following need to hold:

- The inherited and local condition of a style must be compatible (i.e., not inconsistent).
- The local condition must be more specific than the inherited condition (i.e., it must denote a subset of the users/contexts of use denoted by the inherited condition).

The hierarchical verification mechanism of MENTOR checks that such constraints are not violated. Finally, MENTOR also supports verifying that the conditions on the two styles selected as arguments of a design relation are compatible with the type of the relation. This facility is introduced to ensure that the resulting runtime adaptation logic will be semantically sound on the basis of declared design parameters and will not contain ambiguities resulting in problems when applying adaptations (e.g., the decision-making components of the design interface encounters a situation where two styles are declared as incompatible but the conditions on these styles can hold on the same user/context at the same time). The adopted formal verification approach is reported in (Savidis et al., in press).

4.6. Documenting the Adaptation Rationale

Besides verification facilities, the transition from adaptation design to adaptation implementation is supported in MENTOR through the automatic generation of de-

<table>
<thead>
<tr>
<th>Failure</th>
<th>Requirement</th>
<th>Proposed Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexically and/or syntactically incorrect rules</td>
<td>Ensuring rules lexical and syntactic correctness</td>
<td>Lexical and syntactic checker on DMSL condition expressions and automatic generation of DMSL rules</td>
</tr>
<tr>
<td>Adaptation rules are not applicable</td>
<td>Ensuring logical satisfiability of adaptation rules</td>
<td>Formal verification of DMSL condition expressions satisfiability</td>
</tr>
<tr>
<td>Adaptation rules are called in a wrong interaction context</td>
<td>Ensuring hierarchical consistency and incremental specification of style conditions</td>
<td>Formal verification of hierarchical consistency and subsumption of DMSL condition expressions in a polymorphic task hierarchy</td>
</tr>
<tr>
<td>Adaptation rules are ambiguous</td>
<td>Ensuring contextually unambiguous adaptation conditions for designed styles</td>
<td>Formal verification of properties of style conditions according to the type of selected design relation</td>
</tr>
</tbody>
</table>

Note. DMSL = Decision Making Specification Language.
sign documentation for adaptation. This is achieved through the automated collection of all available information in a design case into a textual design documentation report to be subsequently used for several purposes, such as design reviewing and evaluation; interface documentation; and, most important, implementation of the adaptation logic. The design report contains the following:

1. Title: This is the design project’s name.
2. Design parameters: This section lists all defined designed parameters and related values.
3. Profiles: This is a list of the profiles defined in a design case with related conditions.
4. Polymorphic Task Hierarchy: This is a textual representation of the current state of the polymorphic task hierarchy, with each node numbered for easy reference to its properties in the subsequent section.
5. Design artifact properties: This is a numbered list of all artifacts in the hierarchy with their attached properties. Physical design images are also inserted in the text (see left side of Figure 17).
6. Design logic: This is a list of adaptation design decisions, in the form of DMSL rules (see right side of Figure 17), automatically produced by the tool on the basis of the current style conditions and adaptation relations in polymorphic artifacts of the design case. DMSL rules as produced by MENTOR can be directly embedded in the decision-making component of the designed Unified Interface.

5. EVALUATION AND VALIDATION OF MENTOR

MENTOR targets a particularly expert user population, namely, user-interface designers. The evaluation process of the tool therefore has followed a hybrid approach involving a mixture of user satisfaction and expert evaluation methods.

FIGURE 17  The design report generated by MENTOR.
5.1. Evaluation Method

The evaluation method was selected based on the consideration that, besides general heuristics, there are no available heuristics or guidelines for evaluating a design support system from the perspective of the designer. In addition, MENTOR is a unique tool, that is, there are no known existing systems providing equivalent functionality that can be used for comparative assessment. These two facts have determined the selection of empirical evaluation methods. On the other hand, the existence of a high-fidelity prototype made feasible the use of a user-based method to assess the users’ subjective opinion on the perceived usability of the system. The IBM Usability Satisfaction Questionnaires (Lewis, 1995) were adopted for subjective usability measurement in a scenario setting. These questionnaires are available for public use, have been satisfactorily in use for several years now, and are considered as extremely reliable. They include an After-Scenario Questionnaire (ASQ), which is filled in by each participant at the end of each scenario, and a Computer System Usability Questionnaire (CSUQ), which is filled in at the end of the evaluation. The two questionnaires adopt a scale from 1 (strongly agree) to 7 (strongly disagree).

In addition, given the target user group of the tool (i.e., designers, who are by definition expert users), it was decided to combine user satisfaction measurement with expert user-interface evaluation to obtain detailed designers’ comments and suggestions on the MENTOR interface design.

The evaluation was conducted in house and involved seven expert users with substantial experience in user-interface design. All users had at least a university degree in computer science or a related subject. All of them had at least a few years experience in the field of HCI and some basic knowledge but no extensive experience or practice concerning the Unified User Interface Design Method. The user group consisted of four men and three women whose ages ranged from 25 to 35 years ($M_{\text{age}} = 29.5, SD = 3.3$).

The following evaluation procedure was adopted. The group of users was briefly introduced to the main objectives of MENTOR (i.e., the support of adaptation design) and of the evaluation experiments (i.e., the evaluation of user satisfaction and the collection of experts’ comments on the system) and was provided with the related material. Such material included the following:

- A brief introduction to the Unified User Interface Design Method, which underlies the functionality of MENTOR.
- A brief description of the MENTOR functionality and interactive facilities.
- A brief scenario involving two design cases:
  - The design of a predefined Delete File task to be designed for expert and novice users, sighted and blind users, and able-bodied users and users with motor disabilities. Overall, the scenario included the definition of three design parameters with related attributes and values, six user profiles with related conditions, and a polymorphic task hierarchy of 24 design artifacts, out of which 12 styles with the related conditions and 5 polymorphic artifacts with the related adaptation relations among children styles.
- The design of a user task of the designer’s choice, involving importing some design parameters and profiles from the Delete File design case. This was included so that the designers would be able to perform a more extensive exploration of the system’s features.
- The scenario also involved checking all conditions and design relations in the design cases using the automatic verification facilities provided by MENTOR as well as viewing the MENTOR design reports and saving the two cases.

The designers were then requested to perform the tasks in the scenario and fill in the IBM user satisfaction questionnaire as well as a brief evaluation report including their expert evaluation comments on MENTOR and their suggestions for further improvement.

5.2. Evaluation Results

The results of the user satisfaction questionnaire are reported in Table 3 (ASQ) and Table 4 (CSUQ).

5.3. Discussion of the Evaluation Outcomes

According to Table 3, the conduct of the two assigned scenarios appears to have been of medium difficulty, whereas the Delete File scenario appears to have been slightly easier than the free design case. This is probably due to the need of designers to acquire in-depth experience in the application of the Unified User Interface

Table 3: After-Scenario Questionnaire Results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
<th>User 6</th>
<th>User 7</th>
<th>Average</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Delete file</td>
<td>3.3</td>
<td>3.6</td>
<td>3.3</td>
<td>4.0</td>
<td>3.6</td>
<td>3.6</td>
<td>5.0</td>
<td>3.8</td>
<td>0.6</td>
</tr>
<tr>
<td>B: A task of designer’s choice</td>
<td>3.6</td>
<td>4.0</td>
<td>3.3</td>
<td>4.0</td>
<td>3.6</td>
<td>3.6</td>
<td>5.0</td>
<td>3.9</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Note. The range was from 1 (highest) to 7 (lowest).

Table 4: Computer System Usability Questionnaire Results

<table>
<thead>
<tr>
<th></th>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
<th>User 6</th>
<th>User 7</th>
<th>Average</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSUSE</td>
<td>2.60</td>
<td>2.75</td>
<td>2.75</td>
<td>3.60</td>
<td>2.80</td>
<td>2.80</td>
<td>3.00</td>
<td>2.9</td>
<td>0.3</td>
</tr>
<tr>
<td>INFOQUAL</td>
<td>4.30</td>
<td>3.30</td>
<td>3.30</td>
<td>3.60</td>
<td>3.10</td>
<td>3.60</td>
<td>3.50</td>
<td>3.5</td>
<td>0.3</td>
</tr>
<tr>
<td>INTERQUAL</td>
<td>3.30</td>
<td>3.00</td>
<td>2.00</td>
<td>2.60</td>
<td>2.00</td>
<td>1.60</td>
<td>3.30</td>
<td>2.5</td>
<td>0.6</td>
</tr>
<tr>
<td>OVERALL</td>
<td>3.00</td>
<td>3.50</td>
<td>2.50</td>
<td>3.00</td>
<td>2.50</td>
<td>3.00</td>
<td>4.00</td>
<td>3.2</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Note. OVERALL metric provides an indication of the overall satisfaction score, SYSUSE metric provides an indication of the system’s usefulness, INFOQUAL metric is the score for information quality, and INTERQUAL metric is the score for interface quality.
Design Method before developing their own task decompositions and suggests that the learning curve using MENTOR should also be assessed. The most appreciated aspects of the system were found to be its ease of use and overall effectiveness for the purposes of conducting Unified User Interface Design cases (see SYSUSE in Table 4). The users found the system’s interface to be pleasant and intuitive (see INTERQUAL in Table 4). On the other hand, information quality (see INFOQUAL in Table 4) and overall satisfaction (see OVERALL in Table 4) have obtained less positive scores.

More specific comments were provided by the users/experts in their brief evaluation reports, which were provided in addition to the IBM Questionnaire. Based on their user-interface design experience, the following positive characteristics of MENTOR were mentioned by the users/experts:

- Good match between the system functionality and the design of the MENTOR user interface.
- Low cognitive load (to be confirmed through future more formal assessment).
- Use of words, phrases, and concepts familiar to interface designers.
- Self-explanatory user interface, screen display generally containing relevant information.
- Consistent font choices, colors, and sizes; intuitive icons; and other graphical elements.
- Careful design to prevent common problems from occurring in the first place.
- Visibility of objects, actions, and dependencies so that the user does not have to remember information from one part of the tool (e.g., Design Parameters Editor, Profile Editor) when performing tasks in another part (e.g., Polymorphic Task Hierarchy Editor).
- Overall satisfactory system feedback, in particular concerning user input and the recognition, diagnosis, and recovery from possible errors.
- Clear error messages, precisely indicating the problem at hand.

Proposals put forward by the users/experts in their reports toward improving the MENTOR’s user interface included

- Better utilization of white space and placement of internal windows.
- Addition of keyboard short cuts, Tab navigation within tables (e.g., in the Design Parameters Editor), and additional pop-up menus.
- Menu reorganization for better fit of menu options in the context in which they were provided (e.g., the Add option in the Edit menu).

Specific comments regarding the Design Parameters Editor concerned the need for ensuring that errors are avoided when entering value ranges for some attributes (e.g., checking commas between values in enumerated attributes). Concerning the Condition Editor, it was observed that the order of the elements in the interface does not correspond to the order in which tasks are to be performed (first atomic conditions are composed and subsequently operators and parentheses are added), and a reordering of the elements was suggested. A proposed improvement was to
include in dialogues (e.g., in the condition checking feedback) suggestions for correcting errors.

Concerning the Polymorphic Task Hierarchy Editor, the need for copy and paste functions for larger parts of the tree was pointed out; concerning the Properties Editor, it was observed that an editing facility for subtask temporal relations would be helpful.

In general, participants stressed that the availability of a similar tool would be very helpful, in their opinion, in making the Unified User Interface Design Method more easily practiced. However, it was also noted that a certain degree of familiarity with the tool needs to be acquired before effective use in real design cases, particularly in relation to the order of the tasks that the user has to perform, which may not be clear at first glance. This is in part a purposeful characteristic of the tool, reflecting the organization of the Unified User Interface Method, which does not require that design parameters or profiles be defined before any style is created in the Polymorphic Task Hierarchy and its condition edited. However, there are indeed some order constraints on Unified User Interface Design tasks that could be more evident in the MENTOR user interface (e.g., profiles cannot be defined until some design parameters are available). A useful suggestion put forward by four of the users was the addition of a step-by-step wizard to better guide novice users of the tool and act as a tutorial for understanding the tool’s functionality. In general, the need of easily searchable support information (e.g., in the form of Frequently Asked Questions) and of instructional interactions was pointed out.

Furthermore, some of the users had specific requests for additional functionality and system capabilities they would like to see supported in future versions of the system. These mainly concerned the inclusion of collaboration facilities for supporting multiple-user usage of the tool. In general, the user evaluation of MENTOR offered valuable insights into the functional and the interaction characteristics of the system and reinforced the belief that there is an actual need and demand for process-oriented tool supporting the conduct of Unified User Interface Design.

5.4. Validation of MENTOR in Design Case Studies

MENTOR has been validated in a number of design case studies. One of these case studies, concerning the application of the Unified User Interface Design Method in the context of a Health Telematics scenario, is reported in detail in Savidis, Antona, and Stephanidis (2005). More specifically, the scenario addresses personalized access to medical data anytime and from anywhere through the provision of a centralized, continuously updated Web-based repository of electronic health records. In such a context, access to patient health data is envisioned for a variety of purposes in different situations, such as emergencies and medical visits, by users with different roles, and consequently different access rights to the system, and through a variety of platforms, such as the desktop PC and the Wireless Application Protocol phone. The following cases are considered:
- Doctors can have access to the Electronic Health Records (EHR) system through a PC, personal digital assistant (PDA), or mobile phone, viewing all types of data for any patient.
- Ambulance paramedic staff is equipped with a PDA or a mobile phone to perform specific tasks and view patient health data that are critical in emergency situations and for emergency care (e.g., allergies, vaccinations).
- Nurses in the hospital can have access to patient administrative and medication data through a PC or PDA.
- Patients themselves can have access to their data through a PC, PDA, or mobile phone and can opt to access all data or only the most recent (e.g., last week’s blood test results).
- Users in all roles can easily send urgent notifications, including data from the EHR, to other users that may need to be informed (e.g., particular allergy data sent from the ambulance to a doctor in the hospital to obtain advice for emergency treatment, abnormal test results sent from the doctor to the patient or vice versa).

This scenario constitutes a particularly interesting validation context for the Unified User Interface Design Method, as it naturally lends itself to investigating diversity in design parameters and envisaging alternative designs.

Figure 18 illustrates context and platform design parameters definition for the specific design case. Figure 19 shows the polymorphic decomposition of the View HER task in the Polymorphic Task Hierarchy, whereas Figure 20 depicts the adaptation logic in the design report of the case study.

Another example of design case conducted using MENTOR is reported in (Antona, Savidis, & Stephanidis, in press) and focuses on Ambient Intelligence environments. The adopted scenario is an extension of ISTAG scenario n.3 (ISTAG), entitled Carmen, to cater for the case in which the user is blind. The original version of the scenario envisages an ambient intelligence application addressing the issues of traffic, sustainability, and commerce. Interaction in such an environment can
take place, according to various circumstances, through different devices, such as an eFridge in the users’ kitchen and a wearable device called pCom. In the extended scenario, it is assumed that the Ambient Intelligence infrastructure enables users to configure and store their requirements according to specific disabilities (i.e., blindness) for various daytime life situations such as self-care activities, transportations, domestic duties, employment, communication, leisure activities, and social participation in general. Carmen has registered into the system all her requirements and preferences for every situation through a voice recognition system. The information is stored in separate profiles, which can be easily activated through brief voice commands or via text/keyboard insertions. In addition, Carmen’s pCom configuration, in comparison to standard pCom setup, has activated alternative accessibility features supporting nonvisual interaction and con-

**FIGURE 19** Polymorphic decomposition of the View HER task in the Electronic Health Record design case.

**FIGURE 20** The design logic of the Electronic Health Record design case.
trol. Interaction is mainly based on voice commands and sounds as well as on force-feedback and motion-tracking mechanisms. Such features can be used as main preferences by blind and low-vision persons and by normal-sighted individuals as an alternative interaction technique, in situations where constant vision control on the pCom is not possible (e.g., while driving). Figure 21 depicts a part of the Polymorphic Task Hierarchy of the design case and shows the properties of a polymorphic physical style.

Overall, the MENTOR's validation case studies have confirmed its usefulness and its advantages compared with conducting Unified User Interface Design without computational support. The designers who conducted the case studies were able to rapidly acquire familiarity with the Unified User Interface Design Method and with the use of the tool itself, and they expressed the opinion that the tool appropriately reflects and complements the method and significantly simplifies the conduct of polymorphic task decomposition. In particular, the tool has been found helpful in progressively revising and refining the polymorphic task hierarchy of the case studies and in assigning and revising style conditions and design relations. The systematic organization of design cases as supported by MENTOR has helped the designers in articulating in a complete and coherent fashion the polymorphism required by the dimensions of diversity considered in the specific design case. The verification facilities have also been found particularly effective in helping the designer to detect and correct inconsistencies or inaccuracies in the style conditions. Furthermore, the tool has been considered particularly useful in providing the automatic generation of the unified interface adaptation logic. Overall, the design cases have demonstrated that it is possible for designers who are novice in the practice of the Unified User Interface Design Method to quickly manage the conduct of the method and the use of MENTOR and to produce correct and ready-to-implement design specifications for middle-scale design projects. On the other hand, the main problems encountered in using the tool concerned the need for more extensive support in editing the polymorphic task hierarchy (e.g., extended copy and paste functionality for larger parts of the hierarchy) and in making more visible the current completeness status of the design case (completed tasks vs. pending tasks).

**FIGURE 21** A part of the polymorphic task hierarchy in the Ambient Intelligence design case.
6. CONCLUSIONS

In this article, we propose a tool-based approach to the provision of process-oriented support for user-interface design that facilitates the design of user- and context-dependent adaptations, which has been instantiated in an interactive design environment called MENTOR. MENTOR embodies the Unified User Interface Design Method, which has been developed in recent years with the explicit purpose of supporting the design of universally accessible interfaces that exhibit intelligent adaptation behavior and has been applied and validated in important design cases. Such a method has a strong process-oriented perspective and is characterized by a series of features that make it uniquely suited to supporting user-interface design for Universal Access. These include, beside process orientedness, an open-world view of the design domain (i.e., user- and context-oriented factors relevant for design, interaction platforms, languages and artifacts, and possible types of adaptation), a focus on supporting implementation and fostering design reusability and extendibility, and a limited need for formal design representations and modeling. MENTOR provides editing facilities to support the Unified User Interface Design process as well as the design-specification generation facilities. The automated design support facilities offered by MENTOR include the verification of the adaptation logic embedded in design cases and the automatic generation of a design report including the transformation of such adaptation logic into runtime adaptation rules.

MENTOR is intended to constitute a contribution toward addressing the challenges related to design support under a Universal Access perspective, and in particular toward the following:

- Widening and enhancing the practice of design for Universal Access through the provision of a tool that supports the process-oriented conduct of an appropriate design method and facilitates its adoption.
- Improving cost-effectiveness and quality of designing for diversity by computationally supporting the extension and reuse of design cases.
- Offering the concrete opportunity of conducting large-scale design cases by providing a potential repository of design knowledge for further investigation and refinement of the practice of designing for diversity.

These points need to be empirically validated over a longer period of time through the progressive improvement of the tool, its evolution from a research prototype to a publicly available tool, and its concrete impact on the practices of the interface design community as well as the degree to which it will contribute to facilitate and spread the practice of designing for diversity. However, the results of the in-house evaluation experiments, as well as of the validation case studies, anticipate that the practical usefulness of MENTOR is strongly felt by user-interface designers. MENTOR is available at http://www.ics.forth.gr/mentor/. More extensive evaluation of the tool is also planned.

MENTOR is also believed to contribute to the current lively debate in the HCI community concerning the needs and requirements for task-based design tools by introducing in such a context the dimension of (creative) adaptation design and
proposing an approach toward supporting adaptation design in a user-centered, task-base design perspective. It is believed that, in this respect, the most innovative aspect of the tool relies on its focus on “intelligent adaptation” design rather than on intelligent “adaptation design” and on the provision of formal mechanisms for checking the validity of a designed adaptation logic, based on well-known logic techniques, so as to proactively prevent in the design phase possible adaptation logic design errors and to reduce the need of redesigning the adaptation logic once a unified interface has entered the implementation phase.

Future work is planned in two main directions.

- Concrete use of tool in a number of Unified User Interface Design cases, exhibiting different characteristics, and investigation and analysis of resulting design practices are planned, with the aim to provide further empirical evidence for the type of support required in conducting Unified User Interface Design as well as to extract knowledge from design cases (e.g., frequently used combinations of design parameters and profiles, types of adopted design solutions, etc.) so as to facilitate the emergence and subsequent embedding in the tool of prepackaged libraries or “abstract” design cases, as well as the emergence of de facto guidelines or patterns for designing for diversity.
- Going beyond the current concept of process-oriented tool-based support for Unified User Interface Design by investigating the following:
  - Visualization techniques for alternative graphical representations of the polymorphic task hierarchy, targeted toward simplifying and making less cognitively demanding the concrete operation of the hierarchy.
  - Modeling of (a set of) user-interface toolkit artifacts as a “language” for physical design in design cases based on existing toolkits.
  - The interoperation and concurrent use of a process-oriented design support tool such as MENTOR in a wider design support environment comprising tools for working with guidelines, design critique aids, and prototyping tools.

REFERENCES


Winograd, T., Bennett, J., De Young, L., & Hartfield, B. (Eds.). *Bringing design to software*. New York: Addison-Wesley.