A Support Tool for the Design of User Interface Adaptation

Margherita Antona¹, Anthony Savidis¹ and Constantine Stephanidis¹,²

¹ Foundation for Research and Technology - Hellas (FORTH)
Institute of Computer Science
GR-70013 Heraklion, Crete, Greece
[antona, as, cs]@ics.forth.gr

² University of Crete, Department of Computer Science, Greece

Abstract

Automatic user interface adaptation is considered as a mechanism for supporting universal access to novel interaction technologies in the Information Society, and acquires increased importance in the context of ambient intelligence. However, little attention has been paid so far to supporting user interface designers in effectively designing adaptations. This paper presents a tool providing process-oriented support for the design of user interfaces that are automatically adapted during runtime to individual user and usage-context attributes. The tool, named Mentor, aims at effectively facilitating the conduct of adaptation design, and editing facilities for: (i) the definition of the adaptation design space in terms of user- and context-related attributes and related values; (ii) the definition of user and context profiles based on the defined attributes; (iii) task decomposition, including adaptation decomposition into alternative task decompositions or interaction styles where needed; and (iv) properties of interactive artifacts, including adaptation-related properties. Based on the above edited models, the tool is capable of automatically extracting the adaptation logic of the designed interface. Additionally, MENTOR offers automatic verification facilities for the adaptation related information in the edited models. This paper discusses the functionality and user interface of MENTOR, as well as its practical use in a design case in the context of Ambient Intelligence.

1. Introduction

The evolution of Information Society Technologies towards Ambient Intelligence is anticipated to be a complex and multifaceted process, in which a variety of diverse factors will play a critical role. Ambient Intelligence environments will be characterised by computing and networking technologies present everywhere and embedded in everyday objects, which will be interconnected in a web-like structure and form a virtual all-encompassing computing platform. Such a platform will be composed of multiple distributed processing and interactive units, will be capable of monitoring the user and will exhibit different degrees of intelligence (Stephanidis, 2001b). This paradigm-shift will have clear and profound consequences on the type, content and functionality of the emerging products and services, as well as on the way people will interact with them.

Design of interactive artefacts in this context is likely to become a more complex and articulated process, requiring new methods, techniques and tools. On the one hand, a much wider range of individual and context-related requirements than ever before will need to be taken into account in designing interactive artefacts. On the other hand, accessibility and usability issues will acquire fundamental importance for human-centred technological development. Moreover, the context of use is extended to include the physical and social environment and the number and potential impact of relevant factors increase dramatically with respect to conventional computing devices. Therefore, appropriate, personalised, systematically applicable and cost-effective interaction solutions need to be elaborated. Therefore, a proactive approach towards coping with the multiple dimensions of diversity becomes a prerequisite. The notion of Universal Access has the potential to contribute substantially in this respect, as it caters for diversity in every dimension of human-computer interaction. Design for all (or for the broadest possible target population) entails a thorough understanding of the global execution context of tasks which in turn, can be obtained by a conscious effort to gain insight to the changes occurring in a system’s execution context. Therefore, design for all becomes synonymous to designing for diversity or designing to cope with change. In such a context, it is
unrealistic to expect that a single interface design will ensure high-quality interaction for diverse user groups and contexts of use, and Design for All implies the notion of intelligent user interface run-time adaptation, i.e., the capability of automatically adapting to individual user characteristics and contexts of use through the realization of alternative patterns of interactive behaviour (Stephanidis, 2001a). Adaptation refers both to the system’s capability to tailor aspects of its interactive behaviour prior to an interactive session, in anticipation of a user’s requirements (adaptability), as well as to run-time dialogue enhancements on the basis of dynamically acquired and maintained knowledge regarding the user (adaptivity).

Adaptation needs to be “designed into” the system rather than decided upon and implemented a posteriori. Designers should be prepared to cope with large design spaces to accommodate design constraints posed by diversity in the target user population and the emerging contexts of use in the Information Society. For this purpose, methods are needed to allow capturing alternative embodiments of artifacts depicting the diverse contexts of use that may be encountered (i.e., variation in users, platforms, environment), as well as structuring and organising design alternatives in a manner that can be appropriated by suitable user interface development techniques. Producing and enumerating distinct interface designs through the conduct of multiple design processes would be an impractical solution, particularly in the context of Ambient Intelligence environment, since the overall cost for managing in parallel such a large number of independent design processes, and for separately implementing each interface version, would be unacceptable. Instead, a design process is required capable of leading to a single design outcome that appropriately structures multiple designs and their underlying user and context related parameters, therefore facilitating on the one hand the mapping of design to a target software system implementation, and on the other hand the maintenance, updating and extension of design itself.

The Unified User Interface design method has been developed in recent years to facilitate the design of user interfaces with automatic adaptation behavior (Savidis and Stephanidis, 2004a; Savidis, Akoumianakis and Stephanidis, 2001). These efforts have also pointed out the compelling need of making available appropriate support tools for the design process. Following a brief presentation of the Unified User Interface design method, this paper describes MENTOR, a design environment which embodies the Unified User Interface Design method and supports designers in systematically conducting adaptation design. The paper also discusses the application of MENTOR in the design of interactive artifacts in an ambient intelligence environment.

2. Unified User Interface Design

The Unified User Interfaces development framework (Savidis and Stephanidis, 2004b) introduces a novel approach to intelligent adaptation, with the main objective in such a context to ensure that each end-user is provided with the most appropriate interactive experience at run-time.

A Unified User Interface comprises a single (unified) interface specification that exhibits the following properties:

(i) It embeds representation schemes for user- and usage-context- parameters and accesses user- and usage-context- information resources (e.g., repositories, servers), to extract or update such information.

(ii) It is equipped with alternative implemented dialogue patterns (i.e., implemented dialogue artifacts) appropriately associated to different combinations of values for user- and usage-context- related parameters. The need for such alternative dialogue patterns 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.

(iii) It embeds design logic and decision making capabilities that support activating, at run-time, the most appropriate dialogue patterns according to particular instances of user- and usage-context- parameters, and is capable of interaction monitoring to detect changes in parameters.

As a consequence, a unified interface realises user-adapted behaviour (user awareness), i.e., the interface is capable of automatically selecting interaction patterns appropriate to the particular user, as well as usage-context adapted behaviour (usage context awareness), i.e., the interface is capable of automatically selecting interaction patterns appropriate to the particular physical and technological environment. At run-time, the adaptations may be of two types, namely adaptability, i.e., adaptations driven from initial user- and context- information known prior to the initiation of interaction, and adaptivity, i.e., adaptations driven by information acquired through interaction monitoring analysis.
A particularly important aspect of developing Unified User Interfaces concerns their design. The Unified User Interface design method is a hybrid process-oriented design method, encompassing a variety of techniques such as task analysis, abstract design, design polymorphism and design rationale, and enabling the organization of diversity-based design decisions around a single hierarchical structure (Savidis and Stephanidis, 2004a; Savidis, Akoumianakis and Stephanidis, 2001). The process of designing Unified User Interfaces does not lead to a single design outcome, but to a structured design space. It collects and appropriately represents alternative designs, along with the conditions under which each design should be instantiated at run-time (i.e., an adaptation-oriented design rationale). The key elements of the unified user interface design process are the analysis of the space of user- and context-related design parameters, the polymorphic task decomposition process, and the elaboration of an adaptation-oriented design rationale.

The identification of user- and context-oriented design parameters implies the conceptual categorisation of diversity aspects in all relevant dimensions (users, context of use, access terminal / platform), and the identification of the target design parameters for each design case. There is no predefined or fixed set of attribute categories or values, which are chosen as part of the design process. Designers are free to use case-specific design parameters and to experiment with different sets of them. Some examples of relevant factors are general computer-use expertise, domain-specific knowledge, role in an organizational context, motor abilities, sensory abilities, mental abilities, etc. For simplicity, designers may choose to elicit only those attributes from which differentiated design decisions are likely to emerge. Examples of potential context attributes are acoustic noise and light sources, while 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 <User profile, Platform-Context> constitute execution contexts. Each different style should be designed so as to facilitate specific task execution context(s).

The basic representation adopted in Unified User Interface design is called polymorphic task hierarchy, and combines: (a) hierarchical task analysis (Kirwan and Ainsworth, 1992); (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 (Hoare, 1978). Figure 1 provides an example of a polymorphic task hierarchy.

**Figure 1:** An example of Polymorphic Task Hierarchy
In a polymorphic task hierarchy, the root represents design abstractions, while 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, and namely: (i) user tasks, relating to what the user has to do; user tasks are the center of the polymorphic task decomposition process; (ii) 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; and, (iii) physical designs, which concern the interface components on which user actions are to be performed; physical interface structure may also be subject to polymorphism.

The Unified User Interface design process realizes a hierarchical decomposition of tasks, starting from the abstract level, by incrementally specializing in a polymorphic fashion towards 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.

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

- at which points of a task hierarchy polymorphism should be applied, based on the considered (combinations of) user- and usage-context- attributes; and
- how different styles behave at run-time; 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 sub-task 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 the related task, the design targets leading to the introduction of the style, the supported execution context (i.e., a profile based on the defined user- and context-related design parameters), the style properties and the design relationships among competing styles. 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, namely incompatibility (only one of the alternative styles may be present at run-time), compatibility (any of the alternative styles may be present at run-time), substitution (when the second style is activated at run-time, the first should be deactivated) and augmentation (on the presence of a style at run-time, a second style may also be activated).

3. An interactive environment for Unified User Interface design

MENTOR is a prototype support tool for the process of Unified User Interface design. Its functionality include editing facilities for: (i) encoding declarations (signatures) of design parameters attributes and related value spaces; (ii) encoding polymorphic task hierarchies; (iii) creating stereotypes of adaptation conditions, and (iv) attaching information to the artifacts (nodes) in the polymorphic task hierarchy. Additionally, MENTOR provides automated verification mechanisms for the adaptation logic embedded in Unified User Interface design cases, as well as the automated generation of “ready-to-implement” interface specifications. Figure 2 depicts the overall MENTOR interactive environment. The MENTOR’s Design Parameters Editor (upper left window of Figure 2) supports the encoding of user- and context-related design parameters attributes and related value spaces. It displays parameters attributes, type and value ranges. Designers can add any number of new design parameters, and subsequently edit the type and range of values. Three types are allowed, namely enumerated parameters, with value lists as value range, Boolean parameters, with True and False as value range, and integer parameters, with ranges provided through a minimum and a maximum value. Deletion of a design parameter is also possible as long as such a parameter is not used for the current design case. The tool also offers the possibility of importing design parameters from other design cases, in order to support the reusability of previous designs. In the latter case, consistency checking between the parameters already declared in the current design case and the parameters selected for importing is automatically performed by the system.

The Polymorphic Task Hierarchy editor (lower window of Figure 2) allows designers to perform polymorphic task decomposition and encode the results in a hierarchy. The Editor’s main function is to guide the decomposition process by contextualising the available decomposition actions according to the category of design artifact selected in the hierarchy.
Part of the Polymorphic Task decomposition process consists in assigning specific properties to the artifacts in the polymorphic hierarchy. Different categories of artifacts involve different properties. In MENTOR, artifact properties are entered through a Properties Editor (floating window in the lower part of Figure 2), which displays the contextualised properties of the currently selected artifact in the Polymorphic Task Hierarchy Editor. The most important piece of information to be attached to styles concerns the user and context parameter instantiations that define the style appropriateness at run-time. Style conditions are therefore at the heart of Unified User Interface design. In MENTOR, they are encoded as properties of styles and are formulated in a very simple expression language, specifically the condition fragment of DMSL (Savidis and Stephanidis, 2004b) for adaptation decision making in Unified User Interfaces. An example of style condition in DMSL is User.Vision == False, indicating a certain style (for example, a non-visual interaction style) appropriate for blind users.

**Figure 2:** The MENTOR interactive environment

**Figure 3:** The MENTOR Condition Editor

Semantically, conditions are partial descriptions of users or contexts in the space defined by the current design parameters, and refer to sets of users or contexts for which they hold true at run-time. To facilitate the process of defining conditions, MENTOR provides a Condition Editor (Figure 3) which is displayed in the Properties Editor. The Condition Editor supports the easy and syntactically correct editing of conditions by guiding designers to compose conditions. Additionally, MENTOR allows defining stereotypes, i.e., predefined conditions (instantiated profiles of design parameters). Stereotypes can be defined, previously or in parallel with the polymorphic task decomposition, using a Stereotypes Editor (upper right window in Figure 2) that allows associating a DMSL condition (e.g., User.vision == False) with a name (e.g., blind user). In the properties definition phase, the designer can select among the stereotypes defined in the Stereotypes Editor in order to compose style. Automated verification facilities for DMSL conditions are
also included in MENTOR to facilitate designers in the production of a correct and verifiable, ready-to-implement adaptation logic. These include the verification of the lexical and syntactic correctness, as well as the verifiability, of each DMSL expression separately (both in stereotypes and in styles). For example, designers need to avoid that a contradictory condition such as User.vision == False AND User.vision == True will be assigned to a style and propagated to the user interface implementation phase. Additionally, hierarchical relations among styles in the polymorphic task hierarchy are also checked.

Each style in the task hierarchy implicitly inherits the conditions on its ancestor styles, and 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 verification mechanism of MENTOR checks that:

- The inherited and the local condition are compatible (i.e., not inconsistent). For example, designers need to make sure that a condition User.Vision == False is not assigned to a style in a hierarchy branch following a style with the condition User.Vision == True.
- The local condition of a style is more specific than the inherited condition, i.e., it restricts the set of user and context parameters combination for which the description holds with respect the inherited one.

Adaptation relations are formulated in MENTOR as properties of polymorphic artifacts. MENTOR also supports verifying that the conditions on two styles related through a particular relation are compatible with the type of the relation. For example, if two styles are defined as incompatible, their conditions must not be consistent. This checking facility is introduced in order to ensure that the resulting run-time adaptation logic will be semantically sound on the basis of the declared design parameters, and will not contain ambiguities resulting in problems when applying adaptations.

Finally, the documentation of adaptation design is supported in MENTOR 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 importantly, implementation. The design report contains the project’s design parameters and defined stereotypes, a textual representation of the polymorphic task hierarchy, the properties of each designed artifact, and the designed adaptation logic, i.e., a list of adaptation design decisions, in the form of DMSL rules automatically produced by the tool on the basis of the current styles 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.

4. The design of accessible Ambient Intelligence environments in MENTOR

In order to illustrate how MENTOR supports the design user interface adaptations in Ambient Intelligence environments, an example design case has been elaborated based on 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 for example 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 day 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. Additionally, Carmen’s P-com configuration, in comparison to standard P-com set up, has activated alternative accessibility features supporting non visual interaction and control. 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 also by normal sighted individuals, as an

---

1 The extension of ISTAG scenarios under an accessibility perspective is conducted in the context of the European Design for All e-Accessibility Network (EDeAN) – (http://www.eaccessibility.org/) and of the COST219ter Action “Accessibility for All to Services and Terminals for Next Generation Networks” (http://www.tiresias.org/cost219ter/).
The Polymorphic Task Hierarchy of the design case, a portion of which appears in Figure 5 as encoded in the MENTOR Hierarchy Editor, includes various polymorphic artifacts. The depicted tasks concern the shopping part of the scenario, where the ambient intelligence environment allows the user to consult a cooking recipe through the eFridge screen, and to directly select the ingredients to be purchased, to be subsequently ordered on-line. According to the specification of the extended scenario, this shopping task can be performed either through the eFridge or the pCom. As the eFridge is equipped with a large screen, its use is obviously limited to the case that the user is sighted and located in the kitchen. On the other hand, the pCom is also equipped with a much smaller screen, as well as command recognition and voice synthesis capabilities. Therefore, it can used to accomplish the same task in two basic cases: (i) when the user is sighted, located outside the kitchen, and the attention state is visual (i.e., the user can look at the pCom screen), the pCom can be used in a visual interaction mode, (ii) when the user is blind, or the attention state is non visual, voice interaction can be adopted. The latter is independent from the user’s location, as voice interaction is also required in the case that the user is located in the kitchen, but can not look at the screen of the eFridge. The first user subtask in the scenario is to search for the desired recipe. This task only requires physical design, as there are no other subtasks. On the eFridge, this is accomplished by browsing through a list of available recipes on the screen, and selecting one through a touch button, while on the pCom, it is accomplished through search in the visual mode, and voice command in the non visual mode. Therefore, the pCom design style is also polymorphic. The conditions of such styles are as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vision</td>
<td>Boolean</td>
<td>True, False</td>
</tr>
<tr>
<td>AttentionState</td>
<td>Enumerated</td>
<td>Visual, Nonvisual</td>
</tr>
<tr>
<td>Device</td>
<td>Enumerated</td>
<td>eFridge, pCom</td>
</tr>
<tr>
<td>Environment</td>
<td>Enumerated</td>
<td>Kitchen, Road, Car</td>
</tr>
</tbody>
</table>

**Figure 4: Design Parameters of the “Carmen” design case**

**Table 1: Stereotypes for the “Carmen” design case**

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td></td>
</tr>
<tr>
<td>SightedUser</td>
<td>User.Vision == True</td>
</tr>
<tr>
<td>BlindUser</td>
<td>User.Vision == False</td>
</tr>
<tr>
<td>Context</td>
<td></td>
</tr>
<tr>
<td>KitchenEnvironment</td>
<td>Context.Environment == Kitchen</td>
</tr>
<tr>
<td>Other Environment</td>
<td>Context.Environment == Road OR Context.Environment == Car</td>
</tr>
<tr>
<td>eFridge Device</td>
<td>Context.Device == eFridge</td>
</tr>
<tr>
<td>pCom Device</td>
<td>Context.Device == pCom</td>
</tr>
</tbody>
</table>
1) Search Recipe Physical Design eFridge (Unimorphic Physical Design Style)

2) Search Recipe Physical Design pCom (Polymorphic Physical Design Style)
   Condition: Context.Device == pCom

3) Search Recipe Physical Design pCom Select Food (Unimorphic Physical Design Style)

4) Search Recipe Physical Design pCom Voice Input (Unimorphic Physical Design Style)

Image(s): None

Figure 5: The polymorphic task hierarchy of the “Carmen” design case in MENTOR
Styles 1 and 2 are declared as compatible, since it may be the case that interaction through the eFridge and the pCom take place at the same time (e.g., the user look at the eFridge screen and input voice commands through the pCom). Similarly, styles 3 and 4 are also compatible, as it may be the case that the user requires at the same time visual and non visual (multimodal) interaction on the pCom. Figure 6 depicts the results of the satisfiability and hierarchical consistency checking for one of the above styles. The subsequent interaction step is viewing the results of the search. As the polymorphic task hierarchy in Figure 5 shows, recipe viewing is a polymorphic user task, since its dialogue structure needs to be adapted according to different design parameters. In particular, on the large eFridge screen, both the recipe’s ingredients and cooking instructions are displayed at the same time. From the list of ingredients, the user can easily select those that are to be added to the shopping list and ordered on line.

On the other hand, on the pCom, as in the case of the previous task, two interaction styles are designed, namely a visual style and a non-visual one. Given the limited screen space of this wearable device, the user can chose to see only the ingredients or the cooking instructions of the recipe. In case both are requested, they are displayed sequentially. In this case also it is possible to select ingredients for on-line shopping. In the case of the non-visual style, the process of viewing the recipe is similar, with the difference that the ingredients are spoken one by one and the user may, if required, directly add each item to the shopping list through a voice command. The conditions of these task styles are similar to the conditions of the physical styles in the previously mentioned search task. Finally, Figure 7 provides the adaptation logic for the fragment of the design case discussed above, as automatically generated by MENTOR in DMSL. The four rules correspond to the design relations holding between the involved styles, and state the conditions under which each of the designed styles will be instantiated at run-time.
4. Discussion and Conclusions

In the context of Ambient Intelligence, given the intrinsic multiple dimensions of diversity, appropriate design methods and techniques will need to offer adequate means for the design of user interfaces capable of adaptation behaviour, as well as for devising and structuring the underlying adaptation logic of such interfaces. Clearly, in such a complex, articulated and omnipresent technological environment, adaptation needs to be “designed into” the system rather than decided upon and implemented a posteriori. This paper has outlined the main characteristics of MENTOR, an interactive environment for user interface adaptation design. MENTOR embodies the Unified User Interface design method, which has been proposed in recent years as a method to support the design of user interfaces which automatically adapt to factors that impact on their accessibility and usability, such as the abilities and characteristics of different user groups, but also factors related to the context of use and the access technological platforms. MENTOR provides practical integrated support for all phases of Unified User Interface Design, by appropriately guiding the process and structuring the outcomes of design steps through appropriate editing facilities. Additionally, MENTOR provides support for the transition from design to development of Unified User Interfaces through the automated verification mechanisms for the designed adaptation logic, and through the automated generation of “ready-to-implement” interface specifications. An example design case, based on a typical Ambient Intelligence scenario enriched to address accessibility issues, has been presented in order to illustrate the potential application of MENTOR to the design of user interface adaptation in Ambient Intelligence interactive technologies.

In the envisaged scenario, adaptation design is based on a combination of design factors, some of which are typical of Ambient Intelligence as a context of use (e.g., availability of different devices in different environments, availability of wearable devices, different potential attention states of the user in relation to the environment, etc, see, e.g., Stephanidis, 2001b), while others are related to users characteristics and abilities (e.g., user visual abilities) determining accessibility issues. It is believed that methodologies and tools supporting adaptation design will have a significant role to play in facilitating designers in the complex task of shaping the design space of interaction in Ambient Intelligent environment, as adaptation design, and the consequent development of an adaptation logic, is one of the most novel tasks for interaction designers. Additionally, such tools can provide support for the progressive accumulation of design cases and of the related design experience and knowledge, in particular regarding adaptation, by offering means for extending and re-using (parts of) past design cases.

References


