Design for All: Computer Assisted Design of User Interface Adaptation

CONSTANTINE STEPHANIDIS[1,2], MARGHERITA ANTONA[1], ANTHONY SAVIDIS[1,2], NIKOLAOS PARTARAKIS[1], KONSTANTINA DOULGERAKI[1], ASTERIOS LEONIDIS[1]

[1]*Institute of Computer Science*

Foundation for Research and Technology - Hellas (FORTH)

Heraklion, Crete, GR-70013 GREECE

cs@ics.forth.gr

[2]*Department of Computer Science*

University of Crete, Greece

Table of Contents

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 recognised in the recent past in the light of the profound impact that interactive technologies are progressively acquiring on everybody's life and activities, and of the difficulty in developing usable and attractive interactive services and products (e.g., Winograd, 2001). As the Information Society further develops, the issue of Human-Computer Interaction (HCI) design becomes even more prominent when considering the notions of universal access (Stephanidis, 2001a) 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 often been defined as design for diversity, based on the consideration of the several dimensions of diversity that emerge from the broad range of user characteristics, the changing nature of human activities, the variety of contexts of use, the increasing availability and diversification of information, the variety of knowledge sources and services, and the proliferation of diverse technological platforms that occur in the Information Society.

These issues imply an explicit design focus to systematically address diversity, as opposed to afterthoughts or ad hoc approaches, as well as an effort towards reconsidering and redefining the concept of Design for All in the context of HCI (Stephanidis, 2001a). In the emerging Information Society, therefore, Universal Access becomes predominantly an issue of design, and the question arises of how it is possible to design systems that take into account diversity and satisfy the variety of implied requirements. Research work in the past two decades has highlighted a shift of perspective and reinterpretation of HCI design, in the context of Universal access, from current artifact-oriented practices towards a deeper and multidisciplinary understanding of the diverse factors shaping interaction with technology, such as users' characteristics and requirements and contexts of use, and has proposed methods and techniques that enable to proactively take into account and appropriately address diversity in the design of interactive artifacts (Stephanidis, 2001a). In the framework of such efforts, the concept of design for has been reinterpreted and redefined in the domain of HCI. One of the main concepts proposed in such a context as a solution for catering for the needs and requirements of a diverse user population in a variety of context of use is that of automatic user interface adaptation (Stephanidis, 2001b).

Despite the progress that has been made, however, the practice of designing for diversity remains difficult, due to intrinsic complexity of the task, the current limited expertise of designers and practitioners in designing interfaces capable of automatic adaptation, as well as the current limited availability of appropriate supporting tools.

The rationale behind this Chapter is that the wider practice and adoption of an appropriate design method, supported through appropriate tools, has the potential to contribute overcoming the above difficulties. Towards this end, this Chapter, after highlighting the main issues involved in the effort of designing for diversity, briefly describes a design method, the Unified User Interface design method, which has been developed in recent years to facilitate the design of user interfaces with automatic adaptation behaviour (Savidis & Stephanidis, 2009a). Subsequently, the Chapter discusses a series of tools and components which have been developed and applied in various development projects. These tools are targeted to support the design and development of user interfaces capable of adaptation behaviour, and more in particular the conduct and application of the Unified User Interface development approach. Over the years, these tools have demonstrated the technical feasibility of the approach, and have contributed to reducing the practice gap between traditional user interface design and design for adaptation. They have been applied in a number of pilot applications and case studies.

2 Design for All: Overview of approaches, methods and techniques

Universal Access implies the accessibility and usability of Information Society Technologies by anyone, anywhere, anytime, with the aim to enable equitable access and active participation of potentially all citizens in existing and emerging computer-mediated human activities, by developing universally accessible and usable products and services, which are capable of accommodating individual user requirements in different contexts of use and independently of location, target machine, or run-time

environment. The origins of the concept of Universal Access are to be identified in early approaches to accessibility, mainly targeted towards providing access to computer-based applications by users with disabilities.
Subsequently, accessibility related methods and techniques have been generalised and extended towards more generic and inclusive approaches. HCI design approaches targeted to support Universal Access are often grouped under the term of Design for All.

2.1 Reactive vs. proactive strategies

Accessibility in the context of Human Computer Interaction (HCI) refers to the access by people with disabilities to Information and Communication Technologies (ICT). Interaction with ICT may be affected in various ways by the user's permanent, temporary or contextual individual abilities or functional limitations. For example, someone with limited seeing functions will not be able to use an interactive system which only provides visual output, while some one with limited bone or joint mobility or movement functions which affect the upper limbs will encounter difficulties in using an interactive system which only accepts input through the standard keyboard and mouse. Accessibility in the context of HCI aims to overcome such barriers by making the interaction experience of people with diverse functional or contextual limitations as near as possible to that of people without such limitations.
In traditional efforts to improve accessibility, the main direction followed has been to enable disabled users to access interactive applications originally developed for able-bodied users through appropriate adaptations.
Two main technical approaches to adaptation have been followed. The first is to treat each application separately, and take all the necessary implementation steps to arrive at an alternative accessible version - product-level adaptation. Product-level adaptation practically often implies redevelopment from scratch. Due to the high costs associated with this strategy, it is considered as the least favourable option for providing alternative access. The second alternative is to "intervene" at the level of the particular interactive application environment (e.g., MS-Windows) in order to provide appropriate software and hardware technology so as to make that environment alternatively accessible (environment-level adaptation). The latter option

extends the scope of accessibility to cover potentially all applications running under the same interactive environment, rather than a single application.

The above approaches have given rise to several methods for addressing accessibility, including techniques for the configuration of input / output at the level of the user interface, and the provision of Assistive Technologies. Popular Assistive Technologies include screen readers and Braille displays for blind users, screen magnifiers for users with low vision, alternative input and output devices for motor impaired users (e.g., adapted keyboards, mouse emulators, joystick, binary switches), specialized browsers, and text prediction systems).

Despite progress, the prevailing practices aiming to provide alternative access systems, either at the product- or environment-level, have been criticized for their essentially reactive nature (Emiliani, 2009). Although the reactive approach to accessibility may be the only viable solution in certain cases, it suffers from some serious shortcomings, especially when considering the radically changing technological environment, and, in particular, the emerging Information Society Technologies. The critique is grounded on two lines of argumentation. The first is that reactive solutions typically provide limited and low quality access.

The second line of critique concerns the economic feasibility of the reactive approach to accessibility. Reactive approaches, based on a posteriori adaptations, though important to partially solve some of the accessibility problems of people with disabilities, are not viable in sectors of the industry characterized by rapid technological change. By the time a particular access problem has been addressed, technology has advanced to a point where the same or a similar problem re-occurs. The typical example that illustrates this state of affairs is the case of blind people's access to computers. Each generation of technology (e.g., DOS environment, windowing systems and multimedia) caused a new 'generation' of accessibility problems to blind users, addressed through dedicated techniques, such as text-to-speech translation for the DOS environment, off-screen models, and filtering for the windowing systems.

In some cases, adaptations may not be possible without loss of functionality. For example, in the early versions of windowing systems, it was impossible for the programmer to obtain access to certain window functions, such as window management. In subsequent versions, this shortcoming was addressed by the vendors of such products allowing certain adaptations on interaction objects on the screen.

Finally, adaptations are programming-intensive, and, therefore, are expensive and difficult to implement and maintain. Minor changes in product configuration, or the user interface, may require substantial resources to re-build the accessibility features.

From the above, it becomes evident that the reactive paradigm to accessible products and services does not suffice to cope with the rapid technological change and the evolving human requirements. At the same time, the proliferation of interactive products and services in the Information Society, as well as of technological platforms and access devices, brought about the need to reconsider the issue of access under a proactive perspective, resulting in more generic solutions. This entails an effort to build access features into a product starting from its conception, throughout the entire development life-cycle. In the context of the emerging Information Society, therefore, Universal Access becomes predominantly an issue of design, and the question arises of how it is possible to design systems that permit systematic and cost-effective approaches to accommodating all users. Towards this end, the concept of Design for All has been revisited in the context of HCI (Stephanidis et al., 1998; Stephanidis et al., 1999).

In the context of Universal Access, Design for all in the Information Society has been defined as a general framework catering for conscious and systematic efforts to proactively apply principles, methods and tools, in order to develop IST products and services that are accessible and usable by all citizens, thus avoiding the need for a posteriori adaptations, or specialized design. Design for All, or Universal Design, is well known in several engineering disciplines, such as, for example, civil engineering and architecture, with many applications in interior design, building and road construction. In the context of Universal Access, Design for All either subsumes, or is a synonym of, terms such as accessible design, inclusive design, barrier-free design, universal design, etc., each highlighting different aspects of the concept. Through the years, the concept of Design for All has assumed various main connotations:

- Design of interactive products, services and applications, which are suitable for most of the potential users without any modifications. Related efforts mainly aim to formulate accessibility guidelines and standards in the context of international collaborative initiatives (see Section 2.2).

- Design of products which have standardized interfaces, capable of being accessed by specialized user interaction devices (Zimmermann et al., 2002).
- Design of products which are easily adaptable to different users by incorporating adaptable or customizable user interfaces (Stephanidis, 2001b). This entails an effort to build access features into a product starting from its conception, throughout the entire development life-cycle.

2.2 Accessibility Guidelines and de facto Standards

Guidelines play a key role in the adoption of web accessibility and usability by industries and society. In essence, they constitute a rapidly evolving medium for transferring established and de facto knowledge (know-how) to various interested parties.

Concerning accessibility, a number of guidelines collections have been developed (Vanderheiden et al., 1996; Pernice & Nielsen, 2001). In particular, the Web Content Accessibility Guidelines (WCAG) (W3C, 1999) explains how to make Web content accessible to people with disabilities. Web "content" generally refers to the information in a Web page or Web application, including text, images, forms, sounds, etc. WCAG 1.0 provides 14 guidelines that are general principles of accessible design. Each guideline has one or more checkpoints that explain how the guideline applies in a specific area. WCAG foresees 3 levels of compliance, A, AA and AAA. Each level requires a stricter set of conformance guidelines, such as different versions of HTML (Transitional vs. Strict) and other techniques that need to be incorporated into code before accomplishing validation. Further to WCAG 1.0, in December 2008, the W3C announced a new version of the guidelines, targeted to help Web designers and developers to create sites that better meet the needs of users with disabilities and older users. Drawing on extensive experience and community feedback, WCAG 2.0 (W3C, 2008) improves upon WCAG 1.0 and applies to more advanced technologies.

In general, for a website to comply with accessibility guidelines, it should have at least the following characteristics:

- (X)HTML Validation from the W3C for the pages content
- CSS Validation from the W3C for the pages layout
- At least WAI-AA (preferably AAA) compliance with the WAI's WCAG

- Compliance with all guidelines from Section 508 of the US Rehabilitation Act
- Access keys built into the HTML
- Semantic Web Markup
- A high contrast version of the site for individuals with low vision
- Alternative media for any multimedia used on the site (video, flash, audio, etc).

The usage of guidelines is today the most widely adopted process by web authors for creating accessible web content. This approach has proven valuable for bridging a number of barriers faced today by people with disabilities.

Additionally, guidelines constitute de facto standards, as well as the basis for legislation and regulation related to accessibility in many countries (Kemppainen, 2009).

For example, the US government Section 508 of the US Rehabilitation Act (Rehabilitation Act Amendments, 1998), provides a comprehensive set of rules designed to help web designers make their sites accessible.

Unfortunately, however, many limitations arise in the use of guidelines due to a number of reasons. These include the difficulty in interpreting and applying guidelines, which require extensive training. Additionally, the process of using, or testing conformance to, widely accepted accessibility guidelines is complex and time consuming. To address this issue, several tools have been developed enabling the semi automatic checking of html documents. Such tools make easier the development of accessible web content especially due to the fact that the checking of conformance does not rely solely on the expertise of developers. Developers with limited experience in web accessibility can use such tools for evaluating web content and without the need to go through a large number of check – lists.

As a final consideration, guidelines provide a 'one-size-fits-all' approach to accessibility, which, while ensuring a basic level of accessibility for users with various types of disabilities, does not support personalization and improved interaction experience.

2.3 Design for All as User Interface Adaptation Design

In the light of the above, it appears that single artifact-oriented design approaches offer limited possibilities of addressing the requirements posed by Universal Access.

A critical property of interactive artifacts becomes, therefore, their capability for automatic adaptation and personalization (Stephanidis, 2001b).

Methods and techniques for user interface adaptation meet significant success in modern interfaces. Some popular examples include the desktop adaptations in Microsoft Windows XP, offering, for example, the ability to hide or delete unused desktop items. Microsoft Windows Vista and Seven (7) also offer various personalization features of the desktop based on personal preferences of the user, by adding helpful animations, transparent glass menu bars, live thumbnail previews of open programs and desktop gadgets (like clocks, calendars, weather forecast, etc.). Similarly, Microsoft Office applications offer several customizations, such as toolbars positioning and showing/hiding recently used options. However, adaptations integrated into commercial systems need to be set manually, and mainly focus on aesthetic preferences. In terms of accessibility and usability, for instance to people with disability or older people, only a limited number of adaptations are available, such as keyboard shortcuts, size and zoom options, changing color and sound settings, automated tasks, etc.

On the other hand, research efforts in the past two decades have elaborated more comprehensive and systematic approaches to user interface adaptations in the context of Universal Access and Design for All. The Unified User Interfaces methodology was conceived and applied (Savidis & Stephanidis, 2009a) as a vehicle to efficiently and effectively ensure, through an adaptation-based approach, the accessibility and usability of UIs to users with diverse characteristics, supporting also technological platform independence, metaphor independence and user-profile independence. In such a context, automatic UI adaptation seeks to minimize the need for a posteriori adaptations and deliver products that can be adapted for use by the widest possible end user population (adaptable user interfaces). This implies the provision of alternative interface manifestations depending on the abilities, requirements and preferences of the target user groups, as well as the characteristics of the context of use (e.g., technological platform, physical environment). The main objective is to ensure that each end-user is provided with the most appropriate interactive experience at run-time.

The scope of design for diversity is broad and complex, since it involves issues pertaining to context-oriented design, diverse user requirements, as well as adaptable and adaptive interactive behaviors. This complexity arises from the numerous dimensions that are involved, and the multiplicity of aspects in each dimension. In this context, 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. Therefore, designers need accessibility knowledge and expertise. Moreover, user adaptation must be carefully planned, designed and accommodated into the life-cycle of an interactive system, from the early exploratory phases of design, through to evaluation, implementation and deployment.

Therefore, a need arises of providing computational tools which can support the design of user interface adaptation. In the past, the availability of tools was an indication of maturity of a sector and a critical factor for technological diffusion. As an example, Graphical User Interfaces became popular once tools for constructing them became available, either as libraries of reusable elements (e.g., toolkits), or as higher-level systems (e.g., user interface builders and user interface management systems). As design methods and techniques for addressing diversity are anticipated to involve complex design processes and have a higher entrance barrier with respect to more traditional artifact-oriented methods, it is believed that the provision of appropriate design tools can contribute overcoming some of the difficulties that hinder the wider adoption of design methods and techniques appropriate for Universal Access, both in terms of quality and cost, by making the complex design process less resource-demanding. The main objective in this respect is to offer tools which reduce the difference in practice between conventional user interface development and development for adaptation.

Finally, another prominent challenge in the context of Universal Access has been identified as the need of developing large-scale case study applications providing instruments for further experimentation and ultimately improving the empirical basis of the field by collecting knowledge on how design for diversity may be concretely practiced. Such case studies should not only aim to demonstrate technical feasibility, but also to assess the benefits of the overall approach, as well as of the applied methods and tools.

3 Unified User Interfaces

The *Unified User Interface Development* methodology provides a complete technological solution for supporting universal access of interactive applications and services, through a principled and systematic approach towards coping with diversity in the target user requirements, tasks and environments of use (Savidis & Stephanidis, 2009a). A unified user interface comprises a single (unified) interface specification that exhibits the following properties:

1. 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.

2. It is equipped with alternative 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.

3. 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 user interface realizes:

- User-adapted behavior (user awareness), i.e., the interface is capable of automatically selecting interaction patterns appropriate to the particular user.

- Usage-context adapted behavior (usage context awareness), i.e., the interface is capable of automatically selecting interaction patterns appropriate to the particular physical and technological environment.

From a user perspective, a unified user interface can be considered as an interface tailored to personal attributes and to the particular context of use, while from the designer perspective it can be seen as an interface design populated with alternative designs, each alternative addressing specific user- and usage-context- parameter values. Finally, in an engineering perspective, a unified user interface is a repository of implemented dialogue artifacts, from which the most appropriate according to the specific task context are selected at run-time by means of an adaptation logic supporting decision-making.

At run-time, the adaptations may be of two types:

a) adaptations driven from initial user- and context- information known prior to the initiation of interaction, and
b) adaptations driven by information acquired through context and interaction monitoring.

The former behavior is referred to as *adaptability* (i.e., initial automatic adaptation) reflecting the interface's capability to automatically tailor itself initially to each individual end-user in a particular context. The latter behavior is referred to as *adaptivity* (i.e., continuous automatic adaptation), and characterizes the interface's capability to cope with the dynamically changing or evolving user and context characteristics.

The concept of unified user interface is supported by a specifically developed architecture (Savidis & Stephanidis, 2009b). This architecture consists of independent communicating components, possibly implemented with different software methods and tools (see Figure 1). Briefly, a user interface capable of adaptation behavior includes: (i) information regarding user and context characteristics (user and context profile), (ii) a decision making logic and (iii) alternative interaction widgets and dialogues.

INSERT FIGURE 1

The storage location, origin and format of user-oriented information may vary. For example, information may be stored in profiles indexed by unique user identifiers, may be extracted from user-owned cards, may be entered by the user in an initial interaction session, or may be inferred by the system through continuous interaction monitoring and analysis. Additionally, usage-context information, e.g., user location, environment noise, network bandwidth, etc, is normally provided by special-purpose equipment, like sensors, or system-level software. In order to support optimal interface delivery for individual user and usage-context attributes, it is required that for any given user task or group of user activities, the implementations of the alternative best-fit interface components are appropriately encapsulated.

At design time, the design space is captured through a task hierarchy representation which allows for explicitly assigning alternative designs to node elements, called polymorphic task hierarchy (see figure 2). Alternatives designs, call styles, can affect the syntactic level (i.e., alternative task decompositions) or the lexical level (i.e., alternative (i.e., alternative physical designs, such as layout appearances and widgets). Adaptation relations are established among alternative design styles for each node in the hierarchy. These relations define the run-time adaptation behaviour of the user interface, thus providing the adaptation decision-making logic. They include exclusion (two styles are never active at the same time), compatibility (two styles may be active at the same time), substitution (one style is deactivated and the second one is activated), and augmentation (the second style is activated keeping also the first active).

INSERT FIGURE 2

Upon start-up and during runtime, the software interface relies on the particular user and context profiles to assemble the user interface on the fly, collecting and gluing together the constituent interface components required for the particular end-user and usage-context. In this context, runtime adaptation-oriented decision-making is engaged, so as to select the most appropriate interface components for the particular user and context profiles, for each distinct part of the user interface. The role of the decision-making in UI adaptation is to effectively drive the interface assembly process by deciding which interface components need to be selectively activated. The interface assembly process has inherent software engineering implications on the software organization model of interface components. For any component (i.e., part of the interface to support a user activity or task) alternative implemented incarnations may need to coexist, conditionally activated during runtime due to decision-making. In other words, there is a need to organize interface components around their particular task contexts, enabling them to be supported in different ways depending on user and context parameters. This contrasts with traditional non-adapted interfaces in which all components have singular implementations.

The unified user interface development method is not prescriptive regarding how each component is to be implemented (Savidis & Stephanidis, 2009c). For example, the

alternative ways of representing user-oriented information and decision-making mechanisms may be employed. Also, the method does not affect the way designers will create the necessary alternative artifacts (e.g., through prototyping).
Since its beginning, the Unified User Interface development methodology has been accompanied by tools targeted to facilitate its employment. Early tools developed in this context are discussed in details in (Stephanidis, 2001a). The next sections of this Chapter focus on more recent tools which have been applied in a variety of case studies and have proved to contribute towards a more effective and efficient application of the unified user interface concept, with particular focus on design.

4 Tools for the design of user interface adaptations

Tools developed in recent years to support user interface adaptation design include facilities for specifying decision-making rules, adaptation design tools, adaptable widget toolkits for various interaction platforms, and user interface prototyping facilities.

4.1 Decision Making Specification Language

The role of decision-making in user interface adaptation is to effectively drive the interface assembly process by deciding which interface components need to be selectively activated. The Decision Making Specification Language (DMSL) (Savidis et al., 2005) is a rule-based language specifically designed and implemented for supporting the specification of adaptations. DMSL supports the effective implementation of decision-making, and has been purposefully elaborated to be easier for designers to directly assimilate and deploy, in comparison to programming-based approaches using logic-based or imperative-oriented programming languages.
In DMSL, the decision-making logic is defined in independent decision "if…then…else" blocks, each uniquely associated to a particular dialogue context. The individual end-user and usage-context profiles are represented in the condition part of DMSL rules using an attribute values notation. Three types of design parameters values are allowed: (i) enumerated, i.e., values belong to a list of (more than two) strings specified by the designer; (ii) boolean, i.e., values True or False; and (iii) integer, which are 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. The language is equipped with three primitive statements: (a) dialogue, which initiates evaluation for the rule block corresponding to dialogue context value supplied; (b) activate, which triggers the activation of the specified component(s); and (c) cancel, which, similarly to activate, triggers the cancellation of the specified component(s). These rules are compiled in a tabular representation that is executed at run-time. Figure 3 provides an example of DMSL rule. The representation engages simple expression evaluation trees for the conditional expressions.

INSERT FIGURE 3

The decision-making process is performed in independent sequential decision sessions, and each session is initiated by a request of the interface assembly module for execution of a particular initial decision block. In such a decision session, the evaluation of an arbitrary decision block may be performed, while the session completes once the computation exits from the initial decision block. The outcome of a decision session is a sequence of activation and cancellation commands, all of which are directly associated to the task context of the initial decision block. Those commands are posted back to the interface assembly module as the product of the performed decision-making session.

4.2 MENTOR tool for user interface adaptation

The Unified User Interface design is recognised to require a higher initial effort and investment than traditional HCI design approaches, as it involves the identification of relevant design parameters, the design of alternative interface instances, and the delivery of an interface adaptation logic. MENTOR (Antona et al., 2006)) is a support tool for the process of Unified User Interface design, which has been developed in order to address the following objectives:

- 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.
- 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 re-using and extending (parts of) past design cases.

MENTOR targets the community of interface designers and do not assume deep knowledge of the Unified User Interface design method or particular HCI modeling techniques, while, on the other hand, also support designers more experienced in adaptation design in effectively performing their work.

Figure 4 depicts the overall interactive environment of MENTOR, comprising four main editing environments:

INSERT FIGURE 4

- Design Parameters Editor (Figure 4.1). The Design Parameters Editor supports the encoding of design parameters attributes and related value spaces. These constitute the “vocabulary” for defining the “adaptation space” of user interface under design. Parameters can belong to the user domain (i.e., user characteristics), or to the context domain (i.e., characteristics of the context of use and of the interactive platform). The Editor also supports importing existing design parameters, applying the necessary consistency checking.
- Profile Editor (Figure 4.2). User and context profiles can be defined by setting design parameters values in the Profile Editor. Existing profiles can be imported if consistent with current design case.
- Polymorphic Task Hierarchy Editor (Figure 4.3). The Polymorphic Task Hierarchy editor allows designers to perform polymorphic task decomposition and encode the results in a hierarchy. The Editor guides the decomposition process through decomposition steps.
- Properties Editor (Figure 4.4). This Editor allows assigning specific properties to the artifacts in the polymorphic hierarchy. Different categories of artifacts involve different properties. 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 in MENTOR, are formulated in the condition fragment of DMSL. For polymorphic artifacts, adaptation relations between children styles also need to be specified (selecting among incompatibility, compatibility, augmentation and substitution).

Automated verification facilities for DMSL conditions are also included in MENTOR. These include the verification of the lexical and syntactic correctness, as well as the verifiability of each DMSL expression separately. Additionally, hierarchical relations among styles in the polymorphic task hierarchy are also checked. 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. These verification facilities ensure that the resulting run-time adaptation logic is semantically sound, and does not contain ambiguities which could cause problems when applying adaptations.

MENTOR also produces textual documentation of designs which can be used for several purposes, such as reviewing and evaluation, interface documentation, and, most importantly, implementation. The design report contains the project's design parameters and defined profiles, a textual representation of the polymorphic task hierarchy, the properties of each designed artifact, and the designed adaptation logic, in the form of DMSL rules automatically produced by the tool. The DMSL rules produced by MENTOR can be directly embedded in the decision-making component of the designed user interface.

MENTOR has been validated in a number of design case studies, including the design of a unified user interface in the context of a Health Telematics scenario, as well as the design of a shopping cart (Antona et al., 2006). These case studies have confirmed its overall usefulness, and its advantages compared to "paper-based" adaptation design. The designers involved in the case studies were able to rapidly acquire familiarity with the Unified User Interface design method and with the tool itself, and expressed the opinion that the tool appropriately reflects and complements the method, and significantly simplifies the conduct of polymorphic task decomposition. 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 as particularly useful in providing the automatic generation of the DMSL adaptation logic, which, in the case of the Shopping Cart case study, has been directly integrated in the prototype implementation of the component.

4.3 Interaction toolkits

User interface adaptation necessitates alternative versions of interaction artifacts to be created and coexist in the eventual design space. At the lexical level of interaction this can be achieved through software toolkits capable to dynamically deliver an interface instance that is lexically adapted to a specific user in a specific context of use. Such toolkits are essentially software libraries encompassing alternative versions of interaction elements and common dialogues, each version designed in order to address particular values of the user- and usage-context- parameters. The runtime adaptation-oriented selection of the most appropriate version, according to the end-user and usage-context profiles, is the key element in supporting a wide range of alternative interactive incarnations. It should be noted that the presence and management of the alternative versions is fully transparent to toolkit clients. The latter provides the behavior of a smart toolkit capable to adaptively deliver its interaction elements so as to fit the current usage profile.

4.3.1 EAGER

In order to support Unified Web User-Interfaces, the combination of user-centered design, user-interface prototyping and design guidelines is applied together with Unified User-Interface Design. The proposed methodology (Partarakis et al, 2010a) is derived from the Unified User-Interface Software Architecture and is instantiated in the EAGER software toolkit. In particular, EAGER integrates a Design repository of:

- alternative primitive UI elements with enriched attributes (e.g., buttons, links, radios, etc.)
- alternative structural page elements (e.g., page templates, headers, footers, containers, etc.)
- fundamental abstract interaction dialogues in multiple alternative styles (e.g., navigation, file uploaders, paging styles, text entry).

The EAGER Designs Repository is an extensible collection of implemented and ready-to-use alternative interaction elements which are organized around a Polymorphic Task Hierarchy (Savidis & Stephanidis, 2009a). Each alternative element version, called a style following the terminology of (Savidis & Stephanidis, 2009a), is purposefully designed to address the requirements of specific user and context parameter values. Alternative styles have been designed following typical user-centered design, user-interface prototyping and adoption of design guidelines. Additionally, EAGER design alternatives not only integrate current accessibility guidelines, but also provide a suitable approach to personalized accessibility. In this respect, the EAGER Designs Repository can be viewed as encompassing consolidated adaptation design knowledge, thus greatly facilitating designers in the choice of suitable adaptations according to user-related or context-related parameters.

The Designs Repository component of EAGER provides the designs of alternative dialogues controls in a form of abstract interaction objects and task-level polymorphism. For each alternative version, the respective adaptation rationale is also recorded, including the profile parameters which are adaptively addressed.

An example is provided by images. Blind or low vision users are not interested in viewing images, but only in reading the alternative text that describes the image. In order to facilitate blind and low vision users, two design alternatives were produced which are presented in Figure 5.

INSERT FIGURE 5

The text representation of the image simply does not present the image, but only a label with the prefix 'Image:' and followed by the alternative text of the image. The second representation, targeted to users with visual impairments, is same as the first with the difference that, instead of a label, a link is included that leads to the specific image giving the ability of saving the image. In particular, a blind user may not wish to view an image but may wish to save it to a disk and use it properly. In addition to the above, another design was produced that can be selected as a preference by web portal users in which the images are represented as thumbnail bounding the size that holds on the web page. A user who wishes to view the image in normal size may click on it. In Table 1, the design rationale of the alternative images design is presented, including its adaptation logic.

INSERT TABLE 1

EAGER allows Microsoft® .NET developers to create interfaces that conform to the World Wide Web Consortium accessibility guidelines (W3C, 1999), and which are able to adapt to the interaction modalities, metaphors and user interface elements most appropriate to each individual user, according to profile information (user and context). The process of employing EAGER is significantly less demanding in terms of time, experience and skills required from the developer than the typical process of developing Web interfaces for the “average” user. The benefits gained by using the EAGER toolkit lie on a number of dimensions, including:

- The time required for designing a web application and the detail of design information needed.
- The time required for designing the front end of the application to be used by end users.
- The developer effort for setting up the application.

Through EAGER, the complexity of the UI design effort is radically reduced due to the flexibility provided by the toolkit for designing interfaces at an abstract task-oriented level. Therefore, designers are not required to be aware of the low level details introduced in representing interaction elements, but only of the high level structural representation of a task and its appropriate decomposition into sub tasks, each of which represents a basic UI and system function.

On the other hand, the process of designing the actual front end of the application using a mark-up language is radically decreased in terms of time, due to the fact that developers initially have to select among a number of interface components each of which represents a far more complex facility. Additionally, developers do not have to spend time for editing the presentation characteristics of the high level interaction element, due to the internal styling behavior.

The actual process of transforming the initial design into the final Web application using traditional UI controls introduces a lot of coding. On the contrary, when using EAGER the amount of code required is significantly reduced due to the fact that developer has the option to use a number of plug and play controls each of which represent a complex user task. These controls are contained in the advance UI library of EAGER consisting of a total number of 55K pure code lines. Furthermore, the incorporation of EAGER’s higher level elements make the code more usable, more readable and especially safe, due to the fact that each interaction component introduced is designed separately, developed and tested introducing a high level of code reuse, efficiency and safety.

4.3.2 JMorph

The JMorph adaptable widget library (Leonidis et al., 2010) is another example of toolkit inherently supporting the adaptation of user interface components. It contains a set of adaptation-aware widgets designed to satisfy the needs of various target devices –Swing-based components for PC, AWT-based components for Windows Mobile devices. Adaptation is completely transparent to developers, who can use the widgets as typical UI building blocks.

JMorph instantiates a common look and feel across the applications developed using it. The implemented adaptations are meant to address the interaction needs of older users (Leuteritz et al., 2009), and follow specific guidelines which have been encoded into DMSL rules (Savidis et al., 2005). This approach is targeted to novice developers of adaptable user interfaces, as relieves developers from the task of re-implementing or modifying their applications to integrate adaptation-related functionality.

The developed widgets are built in a modular way that facilitates their further evolution, by offering the necessary mechanism to support new features addition and modifications. Therefore, more experienced developers can use their own adaptation rules to modify the adaptation behavior of the interactive widgets.

The library's implementation using the Java programming language ensures the development of portable UIs that can run unmodified with the same look regardless of the underlying Operating System. Apart from OS independence, the proposed framework offers a solution that targets mobile devices running Windows Mobile.

The JMorph library provides the necessary mechanisms to support alternative look and feels either for the entire environment (i.e., skins) or for individual applications.

For that to be achieved, every widget initially follows the general rules to ensure that the common look and feel invariant will be met, and then applies any additional presentation directives declared as "custom" look and feel rules. A "custom" rule could affect either an individual widget (e.g., the OK button that appears in the confirmation dialog of a specific application) or a group of widgets; therefore, entire applications can be fully skinned since their widgets inherently belong to a group defined by the application itself (e.g., all the buttons that belong to a specific application). The look and feel implementation of JMorph is based on the Synth technology (Sun Microsystems, 2010a).

Every adaptable widget in JMorph extends the relevant primitive Java component (i.e., AdaptableButton extends Java's Swing JButton) to provide its typical functionality, while the adaptation-related functionality is exposed via a straightforward API, the AdaptableWidget API. The API declares one main and two auxiliary methods: the adapt and the get(/set)Function methods. Application developers can apply adaptation by simply calling the adapt method. The notion of the Adapt method and the augmented Set / Get attributes methods has been originally proposed and implemented in the context of the PIM language-based generator of multiplatform adaptable toolkits (Savidis et al., 1997). This zero-argument method is the key method of the whole API, as it encapsulates the essential adaptation functionality and every adaptation-aware widget implements it accordingly. The global adaptation process includes firstly the evaluation of the respective DMSL rules that define the appropriate style and size, and then their application through Synth's region matching mechanism.

For a local look and feel to be applied, the adapt method additionally utilizes the function getter method. The function attribute can be set manually by the designer/developer, and is used on the one hand to decide whether and which transformations should be applied, and on the other hand define the group (i.e., all the buttons appear in the Main Navigation bar) or the exact widget (i.e., the OK button in a specific application) where they should be applied utilizing Synth's name matching mechanism.

The adaptable widgets currently implemented in JMorph include label, button, check box, list, scrollbar, textbox, text area, drop-down menu, radio button, hyperlink, slider, spinner, progress bar, tabbed pane, menu bar, menu, menu item, and tooltip. Complex widgets such as date and time entry have also been developed. Adaptable widget attributes include background color/image, widget appearance and dimensions, text appearance, cursor's appearance on mouse over, highlighting of currently selected items or options, orientation options (vertical or horizontal), explanatory tooltips, etc.

Figure 6 shows some of the available widgets. Adaptable attributes for each widget are summarized in Table 2. All widgets in the library also include a text description which allows easy interoperation with speech-based interfaces, thus offering also the possibility to deploy a non visual instance of the developed interfaces.

INSERT FIGURE 6

INSERT TABLE 2

4.4 Adaptive user interface prototyping

Popular user interface builders provide graphical environments for user interface prototyping, usually following a WYSIWYG ("What You See Is What You Get") design paradigm. Available WYSIWYG editors offer graphical editing facilities that allow designers to perform rapid prototyping visually. Such editors may be standalone or embedded in integrated environments (IDEs), i.e., programming environments which allow developing application functionality for the created prototypes directly. Commonly used IDEs are Microsoft Visual Studio, NetBeans, and Eclipse. IDEs are very popular in application development because they greatly simplify the transition from design to implementation, thus speeding up considerably the entire process. However, no currently available tools integrate adaptable widgets, nor provide any support for developing user interface adaptations. Therefore, prototyping alternative design solutions for different needs and requirements using prevalent prototyping tools may become a complex and difficult task if the number of alternatives to be produced is large and no specific support is provided for structuring and managing the design space. In order to facilitate the employment of the JMorph adaptable widget library described in the previous section towards rapid development of adaptable UIs, it has been integrated into the NetBeans GUI Builder (version 8.0, see Figure 7). The result is claimed to be the first and so far unique tool which supports rapid prototyping of adaptable user interfaces, with the possibility of immediately preview adaptation results.

INSERT FIGURE 7

The choice of NetBeans was based on a thorough survey to identify the most suitable available IDEs candidates to incorporate the Adaptable Widget Library into their GUI Builders. NetBeans was preferred to Eclipse, which offers almost equivalent facilities, because it is better supported and more extensible, as its GUI Builder offers the essential mechanisms that facilitate the integration of custom widgets. The library's integration into the NetBeans built-in tool offers prototyping functionalities such as live "UI" preview, as well as automatic application of specific sizing directives according to the OASIS styleguide. Moreover, NetBeans facilitates the implementation of the application's logic associated with the UI, thus offering not only a prototyping tool but a complete framework supporting the entire application development life cycle (design, development and maintenance).

The NetBeans GUI Builder contains a Palette that displays all the available widgets, initially only Java's built-in widgets, while the designers/developers, experienced or not, are familiar with its straightforward 'drag and drop' functionality to add widgets on a "screen". The Palette can only contain widgets that adhere to the JavaBean specification (Sun Microsystems, 2010b). JavaBeans are reusable software components for Java that can be manipulated visually in a builder tool. Practically, they are classes written in the Java programming language conforming to a particular convention.

They are used to encapsulate many objects into a single object (the bean), so that they can be passed around as a single bean object instead of as multiple individual objects. The integration of JMorph into NetBeans was achieved by implementing every AWL widget as a JavaBean.

To prototype a user interface, the designer will create the application's main window, and will add the common containers (e.g., menu panels, status bar, header) by placing AdaptivePanels where appropriate. The necessary widgets (e.g., menu buttons, labels, text fields) will then be dragged from the Palette and dropped into the design area of the builder.

To customize widgets, the typical process is to manually set the relevant attributes for each widget using the designer's "property sheets". To apply the same adjustment to other widgets, one can either copy/paste them or iteratively set them manually. In the adaptation-enabled process, using the function attribute, the process is slightly different. First, one needs to set the function attribute, then define the required style (e.g., colors, images, fonts), and finally to define the rule (in a separate rule file) that maps the newly added style to the specific function. Whenever the same style should be applied, it is sufficient to simply set the function attribute respectively (CSS-like).

In some cases, more radical adaptations are required with respect to widget customization, as the same physical UI design cannot be applied 'as is.

In these cases, alternative dialogues can be designed by creating a container to host the different screens. The JMorph library offers the means to dynamically load different UI elements on demand, providing the functionality through adaptation rules, and utilizing Java's reflection (introspection) capabilities.

The drag and drop selection and placement of widgets follow a conventional WYSIWYG approach. However, in the specific case, What You See Is One Instance of What You Get, as all adaptation alternatives can be produced in the preview mode if the builder by simply setting some user-related variables (e.g., selecting a profile). During preview, a set of sizing rules are automatically applied to ease the design process. The obtained prototypes can easily be used for testing and evaluation purposes.

The result is a tool which offers the possibility of prototyping adaptable interfaces following standard practices, without the need of designing customized widget alternatives, which are included in the adaptable widgets, or to specify adaptation rules (which are predefined). Through the prototyping tool, it is also possible to preview how adaptations are applied for the defined user profiles. However, more expert designers can easily modify the DMSL adaptation rules, which are stored in a separate editable file, and experiment with new adaptations and varying look and feels. Finally, besides appearance adaptations, the overall approach allows to implement more complex forms of adaptation (e.g., dialogue adaptation) by prototyping alternative dialogues and introducing the respective adaptation logic.

5 Prototype applications and case studies

The methods and tools described in the previous sections have been employed over the years in the development of several prototype applications and services in various domains. These efforts demonstrate both the technical feasibility of the adaptation-based approach, and the progress achieved towards simplifying and improving development practices of user interface adaptation.

5.1 AVANTI and PALIO

The AVANTI universally accessible web browser[1] and the PALIO tourist information system[2] (Stephanidis et al., 2010) constitute the first large applications applying the concepts and methods of Unified User Interfaces (see section 3), as well as the first applications of the DMSL language for the implementation of the decision-making component in their architectures (see section 4.1). While AVANTI constituted an adaptable and adaptive content viewing application that can view any type of content,

[1] The AVANTI web browser has been developed in the context of the ACTS AC042 – AVANTI project (see Acknowledgments).
[2] The PALIO tourist information system has been developed in the context of the IST-1999-20656 – PALIO project (see Acknowledgments).

adapted or not, PALIO supported the creation of adaptable and adaptive content that can be viewed with any kind of browser. Thus, the two complemented each other.

The AVANTI browser provides an accessible and usable interface to a range of user categories, irrespective of physical abilities or technology expertise. Moreover it supports various differing situations of use. The end-user groups targeted in AVANTI, in terms of physical abilities, include: (i) "able-bodied" people, assumed to have full use of all their sensory and motor communication "channels"; (ii) blind people; and, (iii) motor-impaired people, with different forms of impairments in their upper limbs, causing different degrees of difficulty in employing traditional computer input devices, such as a keyboard and/or a mouse. In particular, in the case of motor-impaired people, two coarse levels of impairment were taken into account: "light" motor impairments (i.e., users have limited use of their upper limps but can operate traditional input devices or equivalents with adequate support) and "severe" motor impairments (i.e., users cannot operate traditional input devices at all). Furthermore, since the AVANTI system was intended to be used both by professionals (e.g., travel agents) and by general public (e.g., citizens, tourists), the users' experience in the use of, and interaction with, technology was another major parameter that was taken into account in the design of the user interface. Thus, in addition to the conventional requirement of supporting novice and experienced users of the system, two new requirements were put forward: (a) supporting users with any level of computer expertise; and (b) supporting users with or without previous experience in the use of web-based software.

In terms of usage context, the system was intended to be used both by individuals in their personal settings (e.g., home, office), and by the population at large through public information terminals (e.g., information kiosks at a railway station, airport). Furthermore, in the case of private use, the front end of AVANTI was intended to be appropriate for general web browsing, allowing users to make use of the accessibility facilities beyond the context of a particular information system.

Users were also continuously supported as their communication and interaction requirements changed over time, due to personal or environmental reasons (e.g., stress, tiredness, or system configuration). This entailed the capability, on the part of the system, to detect dynamic changes in the characteristics of the user and the context of use (either of temporary or of permanent nature) and cater for these changes by appropriately modifying itself.

The above requirements dictated the development of a new experimental front-end, which would not be based on existing web browser technology, nor designed following traditional techniques oriented to the “typical user”. In fact, the accessibility requirements posed by the user categories addressed in AVANTI could not be met either by existing customizability features supported by commercial web browsers, or through the use of third-party assistive products.

The Unified User Interface development approach was adopted to address the above requirements, as it provides appropriate methodologies and tools to facilitate the design and implementation of user interfaces that cater for the requirements of multiple, diverse end-user categories and usage contexts.

Following the Unified User Interface design method, the design of the user interface followed three main stages: (a) identification of different design alternatives to cater for the particular requirements of the users and the context of use; (b) integration of the designed alternatives into a polymorphic task hierarchy; and (c) development and documentation of the adaptation logic that drives the run-time selection between the available alternatives.

AVANTI also constituted the first application of the DMLS language. Figure 7 shows an example adaptation decision block in the context of AVANTI. Such decision block is targeted to selecting the best alternative interface components for the “link” task context. The interface design relating to this adaptation decision logic is provided in Figure 8.

INSERT FIGURE 8

INSERT FIGURE 9

Building on the results and findings of AVANTI, PALIO set out to address the issue of access to community-wide services by anyone, from anywhere, by proposing a hypermedia development framework supporting the creation of Adaptive Hypermedia Systems. PALIO supported the provision of tourist services in an integrated, open structure, while it constituted an extension of previous efforts, as it accommodated a broader perspective on adaptation and covered a wider range of interactive encounters beyond desktop access, and advanced the current state of affairs by considering novel types of adaptation based on context and situation awareness. The PALIO framework was based on the concurrent adoption of the following concepts: (a) integration of different wireless and wired telecommunication technologies to offer services through both fixed terminals in public places and mobile personal terminals (e.g., mobile phones, PDAs, laptops); (b) location awareness to allow the dynamic modification of information presented (according to user position); (c) adaptation of the contents to automatically provide different presentations depending on user requirements, needs and preferences; (d) scalability of the information to different communication technologies and terminals; and (e) interoperability between different service providers in both the envisaged wireless network and the World Wide Web.

In the context of PALIO, DMSL has been effectively employed not only for user interface adaptation, but also for adaptable information delivery over mobile devices to tourist users. The decision-making process was based on parameters such as nationality, age, location, interests or hobbies, time of day, visit history, and group information (i.e., family, friends, couple, colleagues, etc.). The information model reflected a typical relational database structure, while content retrieval was carried out using XML-based SQL queries. In this context, in order to enable adapted information delivery, instead of implementing hard-coded SQL queries, query patterns have been designed, with specific polymorphic placeholders filled in by dynamically decided concrete sub-query patterns. For instance, as seen in Figure 10, particular data categories or even query operations may be left “open”, with multiple alternatives, depending on runtime content-adaptation decision making.

INSERT FIGURE 10

The experience gained in the development of AVANTI and PALIO has demonstrated the effectiveness of the use of adaptation-based methodologies, techniques and tools towards the achievement of access to the World Wide Web by a wide range of user categories, irrespective of physical abilities or technology expertise, in a variety of contexts of use and through a variety of access devices, going far beyond previous approaches that rely on assistive or dedicated technologies.

Both AVANTI and PALIO make it possible to adopt a stepwise introduction of adaptation, at different stages of development, thus enabling the progressive introduction of complex accessibility features and facilitating the incorporation of new user groups with distinct requirements in terms of accessibility.

5.2 EDeAN web Portal

The portal of the European Design for All e-Accessibility Network (EDeAN)[3] (Partarakis et al., 2010b) was developed as a proof-of-concept by means of the EAGER toolkit (see section 4.3.). As this was the redevelopment of an existing portal, it provided the opportunity to identify and compare the advantages of using EAGER, both at the developer's site, in terms of developer's performance, as well as at the end-user site, in terms of user-experience improvement.

The new EDeAN portal disseminates information about the scope, objectives and outcomes of the EDeAN networking activities. Through the portal public area (Figure 11) a number of facilities can be accessed, such as information about EDeAN, resources from a dedicated resource centre, news and announcements, frequently Asked Questions, statistics regarding the networking activities and surveys for collecting user feedback. The portal area for subscribed users is intended to support the actual networking activities, and therefore provides a number of communication and collaboration facilities.

INSERT FIGURE 11

The users of the portal have the option to access the portal settings and alter them in order to match their personal characteristics and the characteristics of the context of use. A number of parameters can be set, such as Language, Device & display resolution, Assistive technology, Input Device, Disability and Web familiarity. Additionally, in order to allow users to quickly alter their settings, the quick settings option can be used, offering a number of predefined user profiles.

Adaptations can also take place based in interaction preferences. More specifically, interaction preferences settings can alter the interaction elements used for performing fundamental operations, such as browsing content and images or uploading files. The changes made to these settings are propagated to all portal modules. By manually altering these setting the default adaptation logic that occurs based on the user basic setting is enriched.

Finally, adaptations can take place based on accessibility preferences. Custom accessibility includes all the settings that can be altered to enhance the accessibility characteristics of the final user interface. Although each user interface is already

[3] The EDeAN portal has been developed in the context of the IST-CA-033838 - DfA@eInclusion project (see Acknowledgments).

compliant with the W3C accessibility guidelines, theses settings can further enhance the actual system accessibility and the perceived quality of interaction.

5.3 Applications of the JMorph Library

The JMorph adaptable widget library (see section 4.3.2) has been used in the context of a number of development projects, through which it is continuously refined and enhanced.

The first such example was the ASK-IT Home Automation Application[4], which facilitates remote overview and control of home devices through the use of a portable device. The user interface of the application has the ability to adapt itself according to user needs (vision and motor impairments), context of use (alternative display types and display devices) and presence of assistive technologies (alternative input devices). Figure 12 presents an example of adaptation in the screen of the application which supports room selection on a PDA device. In the left part of the figure, the interface displays a color combination, while in the right part a greyscale is used for enhanced contrast.

INSERT FIGURE 12

The version of JMorph used to develop this application included simple graphics and adaptation rules, and the widgets needed to be used programmatically.

Subsequently, the library has been enriched with more simple and complex widgets specifically designed for older users (Leuteritz et al., 2009). Currently, JMorph is being used in the development of the OASIS service suite (Bekiaris & Bonfiglio, 2009), comprising 12 services in three main domains addressing the quality of life of the elderly, namely Independent Living Applications, Autonomous Mobility, and Smart Workplaces Applications[5]. These applications are intended to be available through three different technological platforms, namely tablet PC, PDA and mobile phone. One of such applications is a 5 cards poker game for older users, which can adapt to three different age and visual acuity profiles, as well as to different levels of expertise in poker playing. In this context, the JMorph library has been distributed to a

[4] The ASK-IT Home Automation Application has been developed in the context of the IST-2003-511298 - ASK-IT project (see Acknowledgments).
[5] The OASIS services are currently being developed by various partners in the FP7-ICT-215754 - OASIS project (see Acknowledgments).

pool of universities, research institutions and companies who are in charge of developing the applications.
Another application currently under development is the REMOTE calendar for older users[6], offering functionalities such as to-do-list, medication reminder, nutrition suggestions, daily activities schedule, and other notifications. Figure 12 presents a preliminary prototype of the calendar to-do-list developed using JMorph in the NetBeans IDE.

INSERT FIGURE 13

6 Conclusions

Recent progress in the field of Universal Access and Design for All, i.e., access by anyone, anywhere and anytime to interactive products and services in the Information Society, has highlighted a shift of perspective and reinterpretation of HCI design, from current artifact-oriented practices towards a deeper and multidisciplinary understanding of the diverse factors shaping interaction with technology, such as users' characteristics and requirements and contexts of use, and has proposed solutions for methods, techniques, and codes of practice that enable to proactively take into account and appropriately address diversity in the design of interactive artifacts.
As a consequence, user interface design methodologies, techniques and tools acquire increased importance of in the context of Universal Access, and strive towards approaches that support design for diversity, based on the consideration of the several dimensions of diversity that emerge from the broad range of user characteristics, the changing nature of human activities, the variety of contexts of use, the increasing availability and diversification of information, the variety of knowledge sources and services, and the proliferation of diverse technological platforms that occur in the Information Society. Two main dimensions of such a perspective are its user-oriented focus, targeted towards capturing and collecting the requirements of a diversity of users in a diversity of usage context, and its adoption of intelligent interface adaptation as a technological basis, viewing design as the organisation and structuring

[6] The REMOTE calendar is currently being developed in the context of the AAL - 2008-1-147 - REMOTE project (see Acknowledgments).

of an entire design space of alternatives to cater for diverse requirements. In a Universal Access perspective, adaptation needs to be “designed into” the system rather than decided upon and implemented a posteriori.

Unified User Interface design 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. Despite progress, 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. Towards overcoming such a difficulty, tool support is required for supporting and facilitating adaptation design.

This Chapter has discussed a series of tools and components developed over a period of more than a decade to support and facilitate the conduct of user interface adaptation design. These include:

- A language for the specification of adaptation decision-making (DMSL, see section 4.1)
- An interactive design environment for user interface adaptation (MENTOR, see section 4.2)
- A toolkit supporting the development of adaptable web-based user-interfaces for the .NET platform (EAGER, see section 4.3.1)
- A toolkit of platform-adaptable interaction widgets, implemented in Java, supporting the development of applications for PCs and mobile devices (JMorph, see section 4.3.2).
- A prototyping solution for adaptable user interfaces within the NetBeans IDE (see section 4.4).

Such tools are claimed to have a significant role to play towards widening and improving the practice of Design for All, and ensuring a more effective transition from the design to the implementation phase. They have been used in practice in a series of case studies involving different types of applications for different purposes, contexts and interaction platforms. These extensive case studies have demonstrated the technical feasibility of the overall adaptation-based approach to Universal Access. Additionally, these developments have provided hands-on experience towards improving the usefulness and effectiveness of the developed tools in different phases of the user interface development lifecycle, and in particular design. During these developments, it has progressively become clear that user interface adaptation can be adopted in practice as a result of reducing the gap with mainstream design practices. Ultimately, this amounts to providing transparent solutions which do not require specific adaptation knowledge and support prototyping. Therefore, more recent solutions have gone into the direction of providing ready to use widget toolkits that integrate all the required adaptation knowledge and logic, as well as supporting the view of alternative designs in mainstream development environments. Obviously, however, such solutions, while achieving the objective of simplifying the design of adaptation as far as alternative widget instances are concerned, still require specialized knowledge and mastering of user interface adaptation mechanisms for designing dialogue adaptation at a syntactic or semantic level, as well as for creating new or modifying existing adaptable widgets.

The tools discussed in this Chapter have also proved their usefulness for educational purposes. In particular, DMSL, MENTOR and, more recently, the JMorph library with its accompanying prototyping solution have been used in the context of an advanced Human Computer Interaction course at the University of Crete, with the objective of introducing post-graduate students to the basics of developing self-adapting user interfaces.

As the Information Society further develops, the issue of efficiently designing user interfaces capable of automatic adaptation behavior becomes even more prominent in the context of the next anticipated technological generation, that of Ambient Intelligence environments (see Chapter ++ of this Handbook). Ambient Intelligence provides a vision of the Information Society where humans are surrounded by intelligent intuitive interfaces that are embedded in all kinds of objects and an environment that is capable of recognizing and responding to the presence of different individuals in a seamless, unobtrusive and often invisible. Clearly, Ambient Intelligence environments are intrinsically based on adaptation, and user- and context-awareness, as well as adaptation decision making, become fundamental. Therefore, current research efforts are targeted towards providing tools and facilities to support user interface adaptation design in Ambient Intelligence environments.

Acknowledgments

The work reported in this Chapter has been partially conducted in the context of the following research projects funded by the European Commission:

ACTS AC042 - AVANTI “Adaptive and Adaptable Interactions for Multimedia Telecommunications Applications” (1/9/1995 - 31/8/1998)

IST-1999-20656 - PALIO “Personalised Access to Local Information and Services for Tourists” (1/11/2000 - 30/4/2003)

IST-2003-511298 - ASK-IT “Ambient Intelligence System of Agents for Knowledge-based and Integrated Services for Mobility Impaired users” (1/10/2004 - 31/12/2008)

IST-CA-033838 - DfA@eInclusion “Design for All for eInclusion” (1/1/2007 - 31/12/2009)

FP7-ICT-215754 - OASIS “Open architecture for Accessible Services Integration and Standardisation” (1/1/2008 - 31/12/2011)

AAL - 2008-1-147 - REMOTE “REMOTE” (1/6/2009 - 31/5/2012)

References

Antona, M., Savidis, A., & Stephanidis, C. (2006). A Process–Oriented Interactive Design Environment for Automatic User Interface Adaptation. International Journal of Human Computer Interaction, 20 (2), 79-116.

Bekiaris, E., Bonfiglio, S. (2009). The OASIS Concept. In C. Stephanidis (Ed.) Universal Access in Human-Computer Interaction. Volume 6 of the Proceedings of the 13th International Conference on Human-Computer Interaction (HCI International 2009), San Diego, CA, USA, 19-24 July, pp. 202–209. Berlin Heidelberg: Springer, LNCS.

Emiliani, P.L. (2009). Perspectives on Accessibility: From Assistive Technologies to Universal Access and Design for All. In C. Stephanidis (Ed.), The Universal Access Handbook (pp. 2-1 – 2-18). Boca Raton, FL: Taylor & Francis (ISBN: 978-0-8058-6280-5, 1.034 pages).

Leonidis, A., Antona, M., & Stephanidis, C. (2010, to appear). Rapid Prototyping of Adaptable User Interfaces.

Leuteritz, J.-P., Widlroither, H., Mourouzis, A., Panou, M., Antona, M., Leonidis, S. (2009). Development of Open Platform Based Adaptive HCI Concepts for Elderly Users. In C. Stephanidis, (Ed.), Universal Access in Human-Computer Interaction - Intelligent and Ubiquitous Interaction Environments. – Volume 6 of the Proceedings of the 13th International Conference on Human-Computer Interaction (HCI International 2009), San Diego, CA, USA, 19-24 July, pp. 684-693. Berlin Heidelberg: Lecture Notes in Computer Science Series of Springer (LNCS 5615, ISBN: 978-3-642-02709-3).

Kemppainen, E., Kemp, J.D., & Yamada, H. (2009). Policy and Legislation as a Framework of Accessibility. In C. Stephanidis, (Ed.) The Universal Access Handbook (pp 53-1 – 53-16). Boca Raton, FL: Taylor & Francis (ISBN: 978-0-8058-6280-5, 1.034 pages).

Partarakis, N., Doulgeraki, C., Antona, M. & Stephanidis, C. (2010a). Designing Web-based Services. In G. Salvendy & W. Karwowski (Eds), Introduction to Service Engineering (pp. 447-487). Hoboken, NJ, USA: John Wiley.

Partarakis, N., Doulgeraki, C., Antona, M. & Stephanidis, C. (2010b). The Development of Web-based Services. In G. Salvendy & W. Karwowski (Eds), Introduction to Service Engineering (pp. 502-532). Hoboken, NJ, USA: John Wiley.

Pernice, K., J, Nielsen (2001). Beyond ALT Text: Making the Web Easy to Use for Users with Disabilities. Nielsen Norman Group Report. http://www.nngroup.com/reports/accessibility

Savidis, A., Stephanidis, C., Akoumianakis, D. (1997). Unifying Toolkit Programming Layers: a Multi-Purpose Toolkit Integration Module. In Proceedings of the Fourth International Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS), 1997, (pp. 177-192), Springer

Savidis, A., & Stephanidis, C. (2009a). Unified Design for User Interface Adaptation. In C. Stephanidis (Ed.), The Universal Access Handbook (pp. 16-1 – 16-17). Boca Raton, FL: Taylor & Francis (ISBN: 978-0-8058-6280-5, 1.034 pages).

Savidis, A., & Stephanidis, C. (2009b). A Unified Software Architecture for User Interface Adaptation. In C. Stephanidis (Ed.), The Universal Access Handbook (pp. 21-1 – 21-17). Boca Raton, FL: Taylor & Francis (ISBN: 978-0-8058-6280-5, 1.034 pages).

Savidis, A., & Stephanidis, C. (2009c). Software Requirements for Inclusive User Interfaces. In C. Stephanidis (Ed.), The Universal Access Handbook (pp. 18-1 – 18-25). Boca Raton, FL: Taylor & Francis (ISBN: 978-0-8058-6280-5, 1.034 pages).

Savidis, A., Antona, M., Stephanidis, C. (2005). A Decision-Making Specification Language for Verifiable User-Interface Adaptation Logic. International Journal of Software Engineering and Knowledge Engineering, 15 (6), 1063-1094.

Shneiderman, B., (2000). Pushing human-computer interaction research to empower every citizen. Universal Usability. Communication of the ACM, May 2000/Vol. 43, No. 5, 85-91.

Stephanidis, C., Paramythis, A., & Savidis, A. (2010, to appear). Developing Adaptive Interfaces for the Web. In R. Proctor & K. Vu (Eds.), Handbook of Human Factors in Web Design (2nd edition), Chapter 14. CRC Press.

Stephanidis, C. (2001a). User Interfaces for All: New perspectives into Human-Computer Interaction. In C. Stephanidis (Ed.), User Interfaces for All -

Concepts, Methods, and Tools (pp. 3-17). Mahwah, NJ: Lawrence Erlbaum Associates (ISBN 0-8058-2967-9, 760 pages).

Stephanidis, C. (2001b). Adaptive techniques for Universal Access. User Modelling and User Adapted Interaction International Journal, 10th Anniversary Issue, 11 (1/2), 159-179.

Stephanidis, C. (Ed.), Salvendy, G., Akoumianakis, D., Arnold, A., Bevan, N., Dardailler, D., Emiliani, P.L., Iakovidis, I., Jenkins, P., Karshmer, A., Korn, P., Marcus, A., Murphy, H., Oppermann, C., Stary, C., Tamura, H., Tscheligi, M., Ueda, H., Weber, G., and Ziegler, J. (1999). Toward an Information Society for All: HCI challenges and R&D recommendations. International Journal of Human-Computer Interaction, 11 (1), 1-28.

Stephanidis C. (Ed.), Salvendy, G., Akoumianakis, D., Bevan, N., Brewer, J., Emiliani, P.L., Galetsas, A., Haataja, S., Iakovidis, I., Jacko, J., Jenkins, P., Karshmer, A., Korn, P., Marcus, A., Murphy, H., Stary, C., Vanderheiden, G., Weber, G., & Ziegler, J. (1998). Toward an Information Society for All: An International R&D Agenda. International Journal of Human-Computer Interaction, 10 (2), 107-134.

Sun Microsystems. (2010a). The Synth Architecture. http://java.sun.com/docs/books/tutorial/uiswing/lookandfeel/synth.html

Sun Microsystems (2010b). JavaBeans Concept. http://java.sun.com/docs/books/tutorial/javabeans/whatis/index.html

The Rehabilitation Act Amendments, Section 508 (1998). http://www.section508.gov/

Winograd, T. (2001). From Programming Environments to Environments for Designing. In C. Stephanidis (Ed.), User Interfaces for All - Concepts, Methods, and Tools (pp. 165-181). Mahwah, NJ: Lawrence Erlbaum Associates (ISBN 0-8058-2967-9, 760 pages).

Vanderheiden, G., Chisholm, W., and Ewers, N. (1996). Design of HTML pages to increase their accessibility to users with disabilities, Strategies for today and tomorrow. Technical Report, Trace R&D Centre, University of Wisconsin - Madison, May 1996.

W3C. Web Content Accessibility Guidelines 1.0, W3C Recommendation, May 1999. http://www.w3.org/TR/WCAG10/.

W3C. Web Content Accessibility Guidelines 2.0, W3C Recommendation, December 2008. http://www.w3.org/TR/WCAG20/

Zimmermann, G., Vanderheiden, G., & Gilman, A. (2002). Universal Remote Console -Prototyping for the Alternate Interface Access Standard. In N. Carbonell & C. Stephanidis (Eds.), Universal Access: Theoretical Perspectives, Practice and Experience - Proceedings of the 7th ERCIM UI4ALL Workshop (pp. 524-531). Berlin, Heidelberg: Springer.

Figures

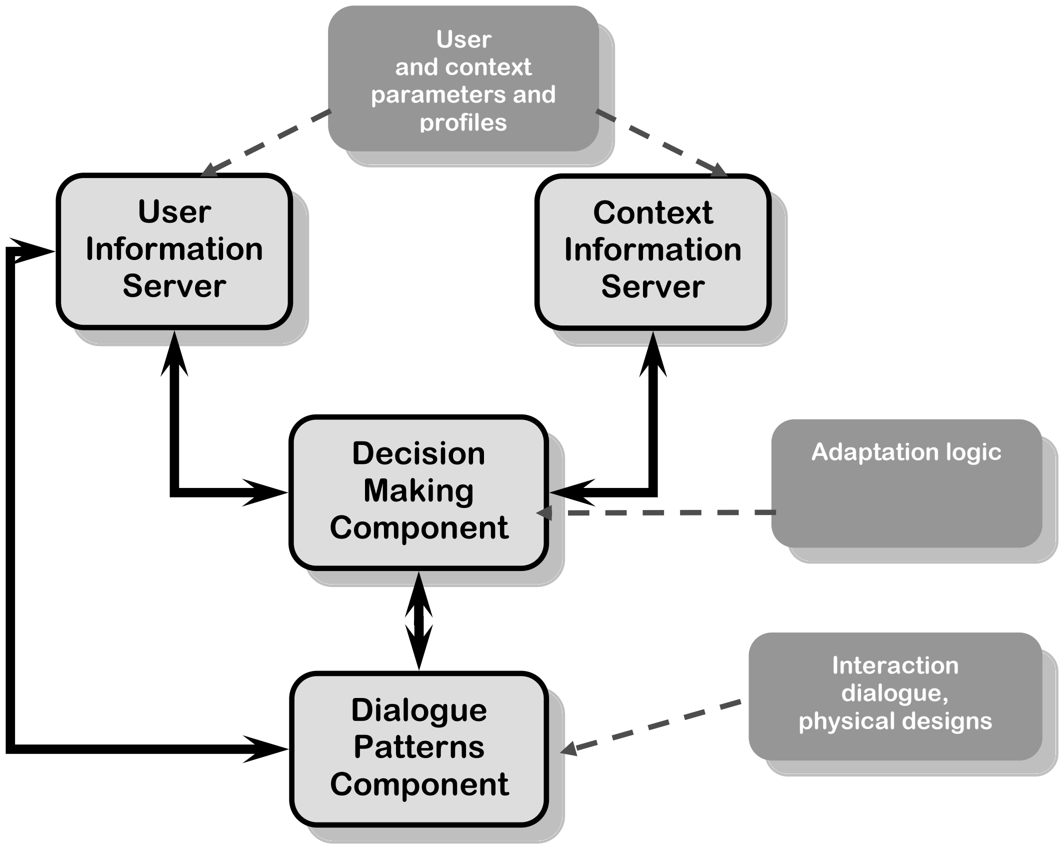

User and context parameters and profiles
User Information Server
Context Information Server
Decision Making Component
Adaptation logic
Dialogue Patterns Component
Interaction dialogue, physical designs

Figure 1. The Unified User Interface Architecture

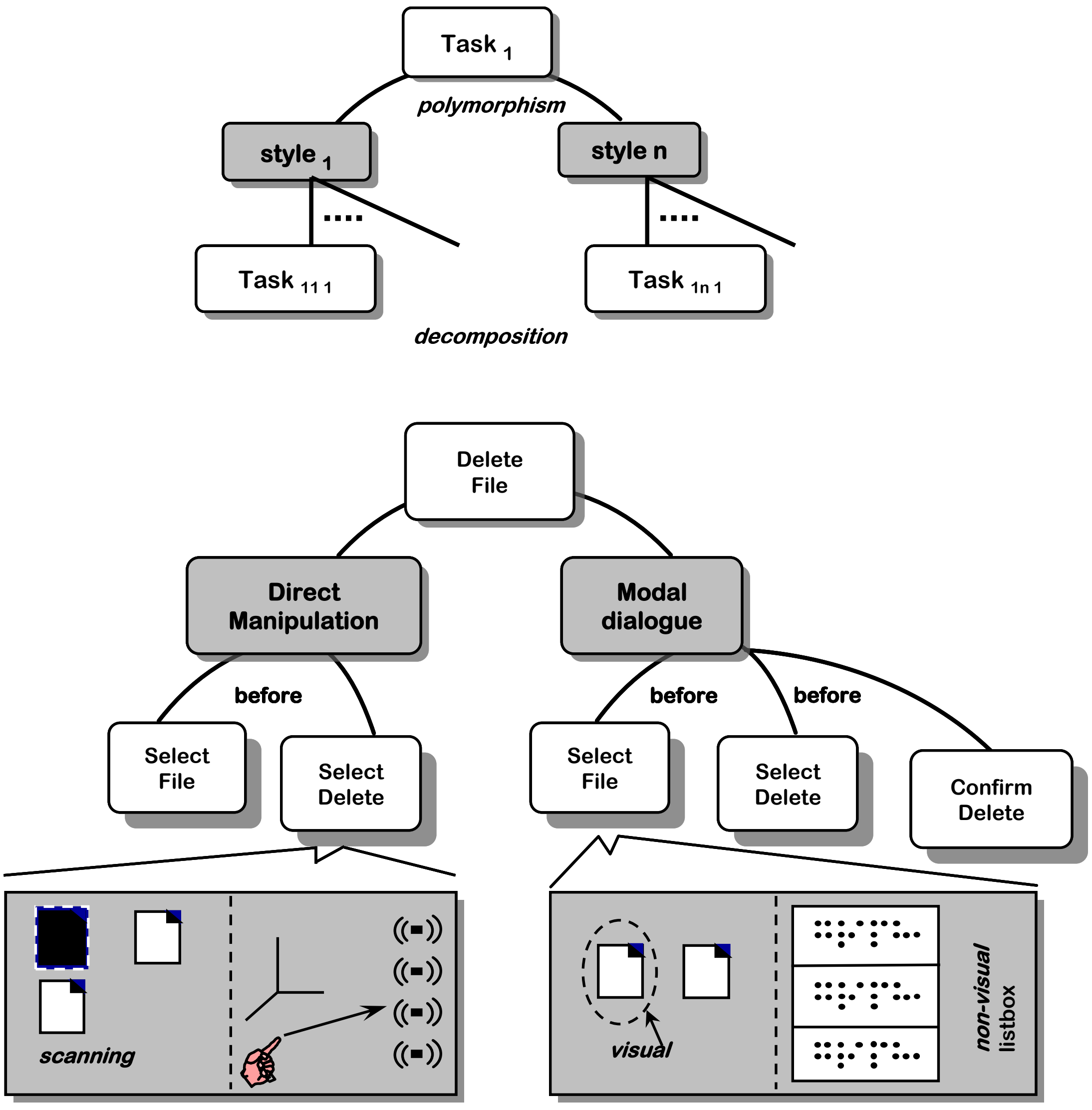
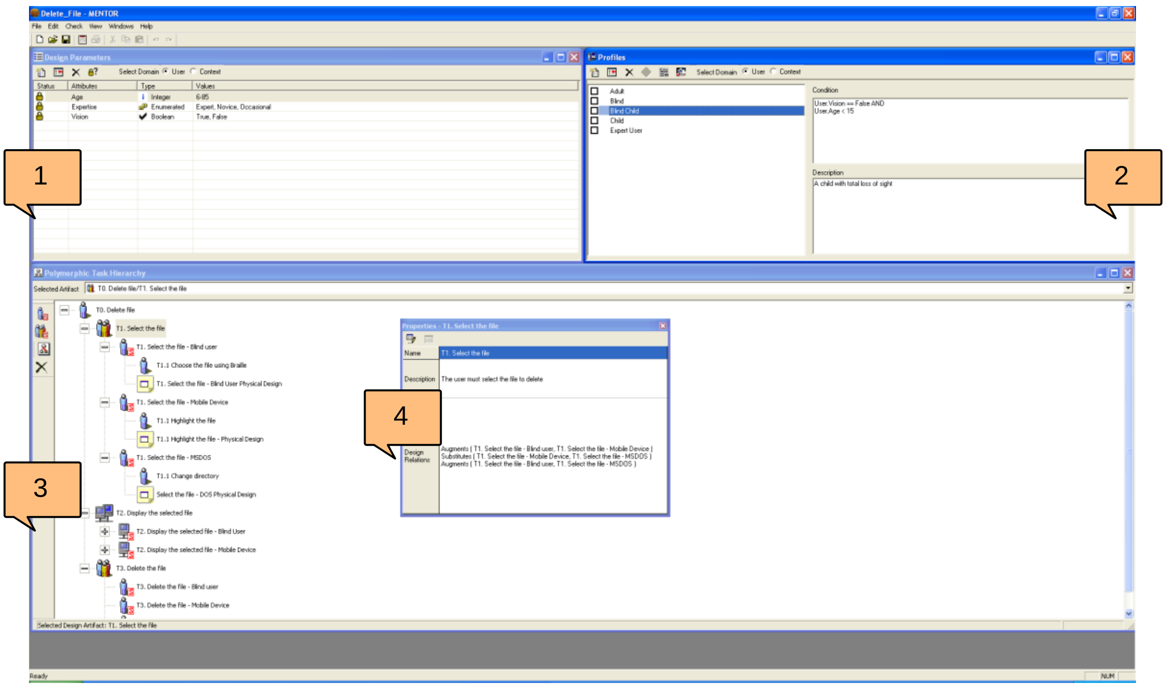


Figure 2. An example of polymorphic task hierarchy

1	If	[Elderly user’s age = 1 or 2 or 3] or [Elderly user’s life situation = 2 or 3] or [Elderly user’s computer literacy level = 0] or [Vision impairment = 1 or 2 or 3]
	Then	Resolution 640*480 pixels
2	If	[Elderly user’s life situation =1] or [Elderly user’s computer literacy level = 1]
	Then	Resolution 800*600 pixels

Figure 3. An example of DMSL rule


Delete_File - MENTOR
File Edit Check View Windows Help
Design Parameters
Profiles
Polymorphic Task Hierarchy
Properties - T1. Select the file
Ready
NUM
1
2
3
4

Select Domain (● User) ○ Context

Status	Attributes	Type	Values
🔒	Age	Integer	6-85
🔒	Expertise	Enumerated	Expert, Novice, Occasional
🔒	Vision	Boolean	True, False

Select Domain (● User) ○ Context

- ☐ Adult
- ☐ Blind
- ☐ Blind Child
- ☐ Child
- ☐ Expert User

Condition

User.Vision == False AND
User.Age < 15

Description

A child with total loss of sight

Selected Artifact: T0. Delete file/T1. Select the file

```
T0. Delete file
    T1. Select the file
        T1. Select the file - Blind user
            T1.1 Choose the file using Braille
            T1. Select the file - Blind User Physical Design
        T1. Select the file - Mobile Device
            T1.1 Highlight the file
            T1.1 Highlight the file - Physical Design
        T1. Select the file - MSDOS
            T1.1 Change directory
            Select the file - DOS Physical Design
    T2. Display the selected file
        T2. Display the selected file - Blind User
        T2. Display the selected file - Mobile Device
    T3. Delete the file
        T3. Delete the file - Blind user
        T3. Delete the file - Mobile Device
```

Selected Design Artifact: T1. Select the file

Name	T1. Select the file
Description	The user must select the file to delete
Design Relations	Augments (T1. Select the file - Blind user, T1. Select the file - Mobile Device) Substitutes (T1. Select the file - Mobile Device, T1. Select the file - MSDOS) Augments (T1. Select the file - Blind user, T1. Select the file - MSDOS)

Figure 4. The overall MENTOR interactive environment: (1) Design Parameters Editor; (2) Profile Editor; (3) Polymorphic Task Hierarchy Editor; (4) Properties Editor

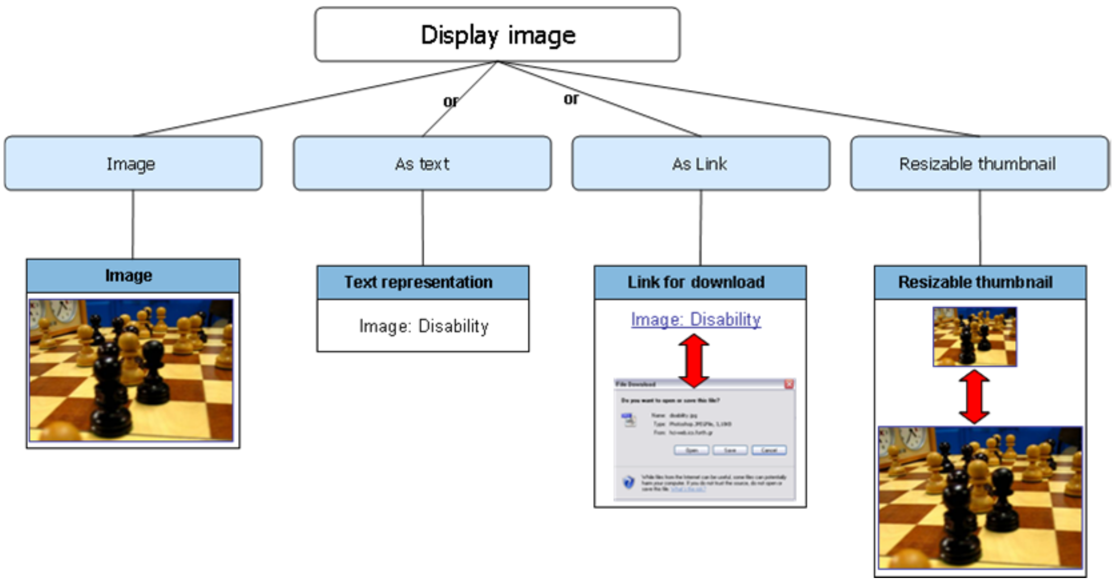


Figure 5. Alternative image hierarchy in EAGER (from Partarakis et al., 2010a)

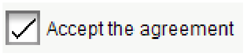
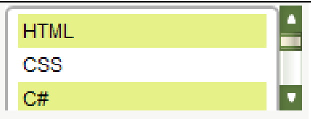
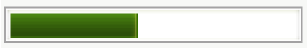
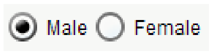
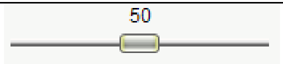
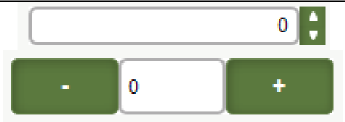
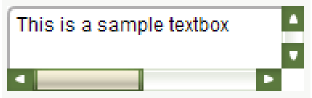
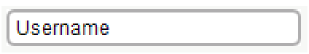
Check button	
Listbox	
Progress bar	
Radio button	
Slider	
Spinner	
Textbox	
Text field	

Figure 6. Examples of adaptable widgets

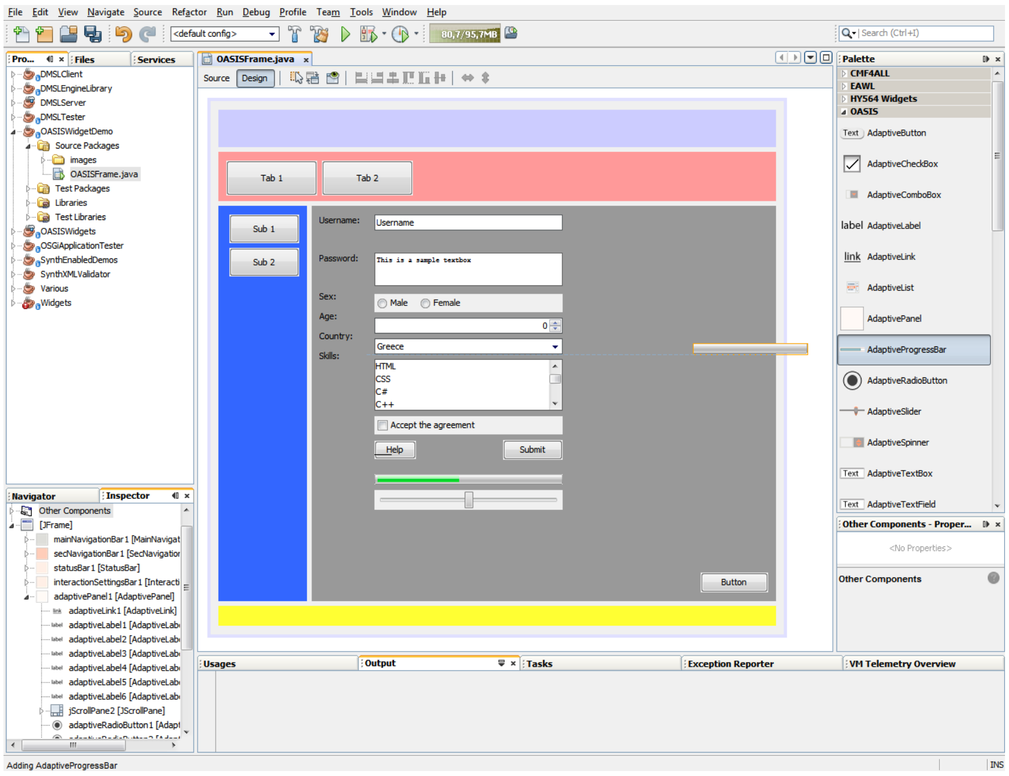


Figure 7. Using the Adaptable Widget Library integrated in the NetBeans IDE

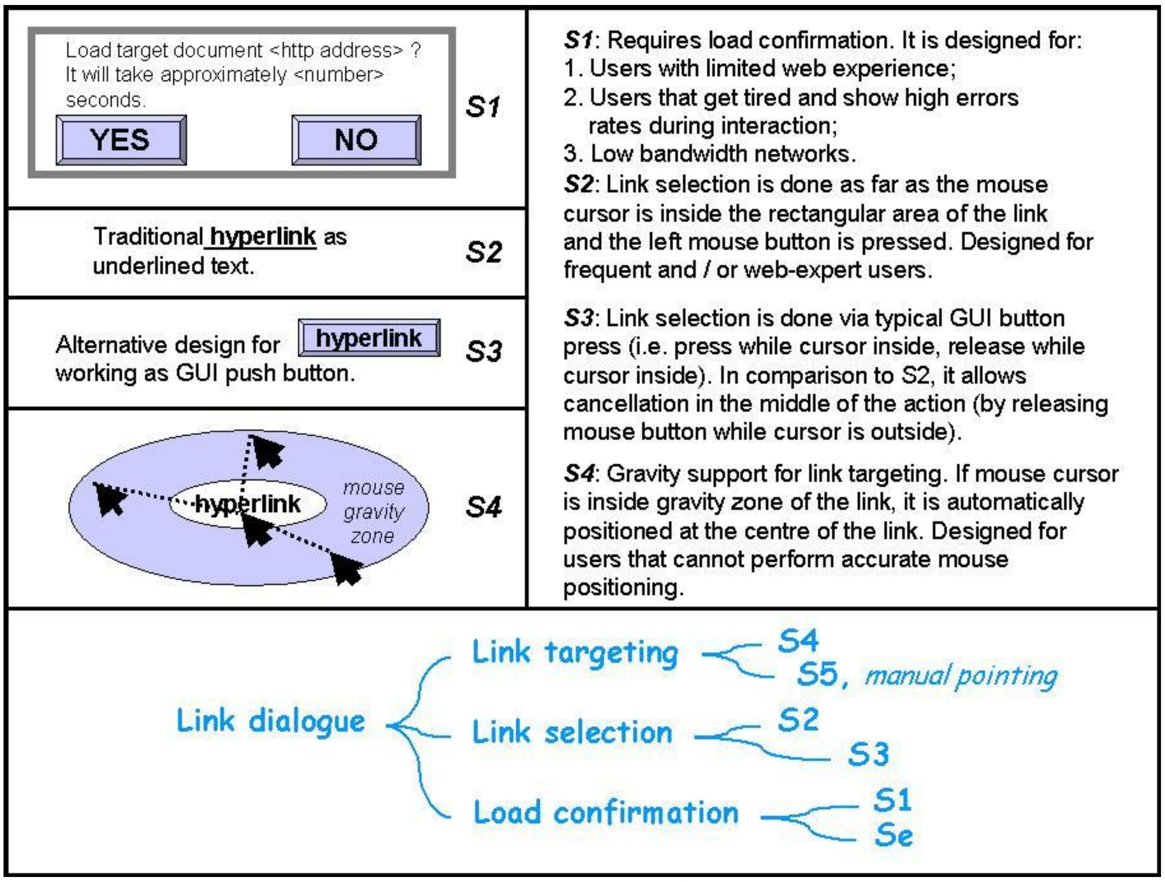
```
taskcontext link  [
       evaluate linktargeting;
       evaluate linkselection;
       evaluate loadconfirmation;
]

taskcontext linktargeting [
       if (user.abilities.pointing == accurate) then
              activate "manual pointing";
       else
              activate "gravity pointing";
]

taskcontext linkselection [
       if (user.webknowledge in {good, normal}) then
              activate "underlined text";
       else
              activate "push button";
]

taskcontext loadconfirmation [
       if (user.webknowledge in {low, none} or context.net==low) then
              activate "confirm dialogue";
       else
              activate "empty";
]
```

Figure 8. DMSL decision block for adaptation of links in the AVANTI browser (from Savidis et al., 2005)


Load target document <http address> ?
It will take approximately <number> seconds.
YES
NO
S1
Traditional hyperlink as underlined text.
S2
Alternative design for working as GUI push button.
hyperlink
S3
hyperlink
mouse gravity zone
S4
Link dialogue
Link targeting
S4
S5, manual pointing
Link selection
S2
S3
Load confirmation
S1
Se

S1: Requires load confirmation. It is designed for:
1. Users with limited web experience;
2. Users that get tired and show high errors rates during interaction;
3. Low bandwidth networks.

S2: Link selection is done as far as the mouse cursor is inside the rectangular area of the link and the left mouse button is pressed. Designed for frequent and / or web-expert users.

S3: Link selection is done via typical GUI button press (i.e. press while cursor inside, release while cursor inside). In comparison to S2, it allows cancellation in the middle of the action (by releasing mouse button while cursor is outside).

S4: Gravity support for link targeting. If mouse cursor is inside gravity zone of the link, it is automatically positioned at the centre of the link. Designed for users that cannot perform accurate mouse positioning.

Figure 9. Link adaptation design in AVANTI (from Savidis et al., 2005)

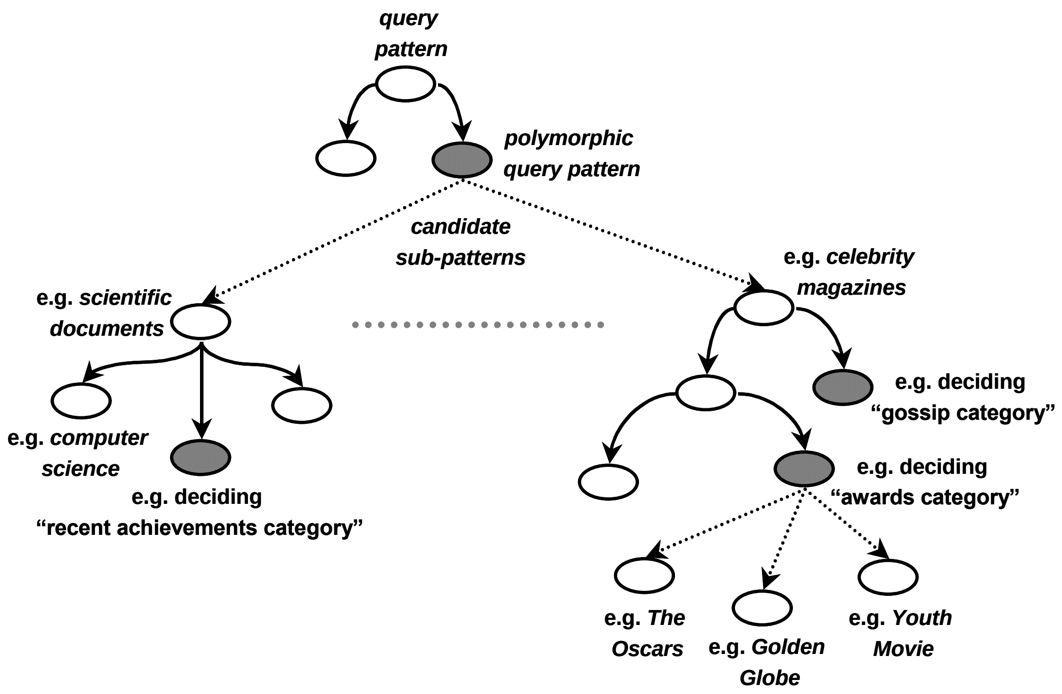
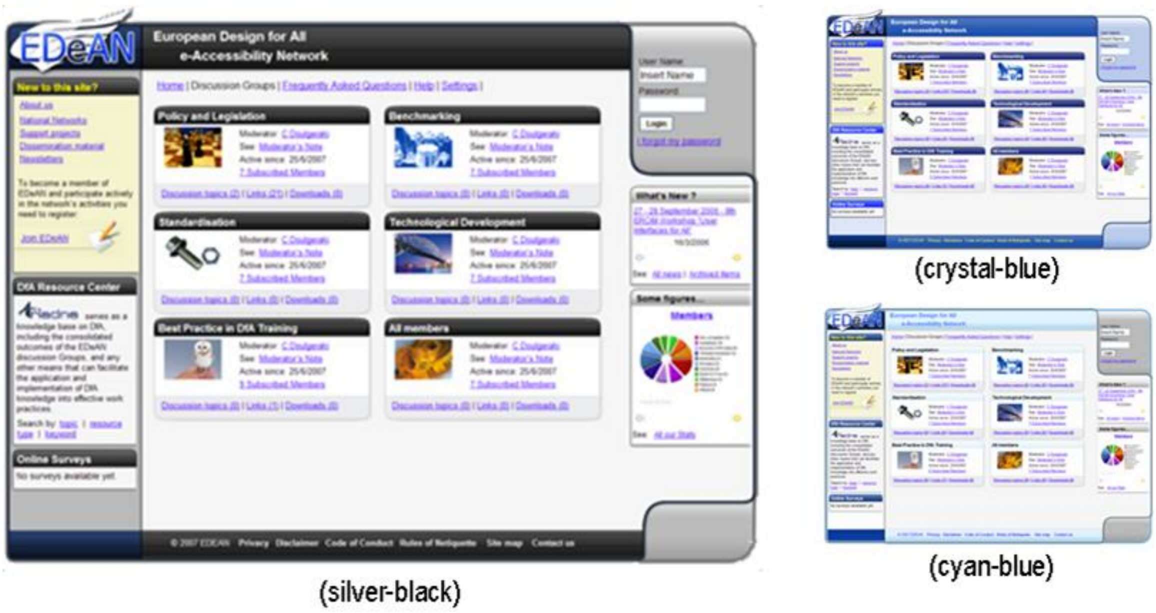

query pattern
polymorphic query pattern
candidate sub-patterns
e.g. scientific documents
e.g. celebrity magazines
e.g. deciding "gossip category"
e.g. computer science
e.g. deciding "recent achievements category"
e.g. deciding "awards category"
e.g. The Oscars
e.g. Golden Globe
e.g. Youth Movie

Figure 10. Query adaptation using DMSL in PALIO (from Savidis et al., 2005)



European Design for All
e-Accessibility Network

Home | Discussion Groups | Frequently Asked Questions | Help | Settings |

New to this site?
About us
National Networks
Support projects
Dissemination material
Newsletters

To become a member of EDeAN and participate actively in the network's activities you need to register.

Join EDeAN

DfA Resource Center
Ariadne serves as a knowledge base on DfA, including the consolidated outcomes of the EDeAN discussion Groups, and any other means that can facilitate the application and implementation of DfA knowledge into effective work practices.
Search by: topic | resource type | keyword

Online Surveys
No surveys available yet.

Policy and Legislation
Moderator: C.Doulgerakis
See: Moderator's Note
Active since: 25/6/2007
7 Subscribed Members
Discussion topics (2) | Links (21) | Downloads (0)

Benchmarking
Moderator: C.Doulgerakis
See: Moderator's Note
Active since: 25/6/2007
7 Subscribed Members
Discussion topics (0) | Links (0) | Downloads (0)

Standardisation
Moderator: C.Doulgerakis
See: Moderator's Note
Active since: 25/6/2007
7 Subscribed Members
Discussion topics (0) | Links (0) | Downloads (0)

Technological Development
Moderator: C.Doulgerakis
See: Moderator's Note
Active since: 25/6/2007
7 Subscribed Members
Discussion topics (0) | Links (0) | Downloads (0)

Best Practice in DfA Training
Moderator: C.Doulgerakis
See: Moderator's Note
Active since: 25/6/2007
9 Subscribed Members
Discussion topics (0) | Links (1) | Downloads (0)

All members
Moderator: C.Doulgerakis
See: Moderator's Note
Active since: 25/6/2007
7 Subscribed Members
Discussion topics (0) | Links (0) | Downloads (0)

User Name
Insert Name
Password
Login
I forgot my password

What's New ?
27 - 28 September 2006 - 9th ERCIM Workshop "User Interfaces for All"
16/3/2006
See: All news | Archived items

Some figures...
Members
See: All our Stats

© 2007 EDEAN Privacy Disclaimer Code of Conduct Rules of Netiquette Site map Contact us

Figure 12. The main page of the EDeAN web portal

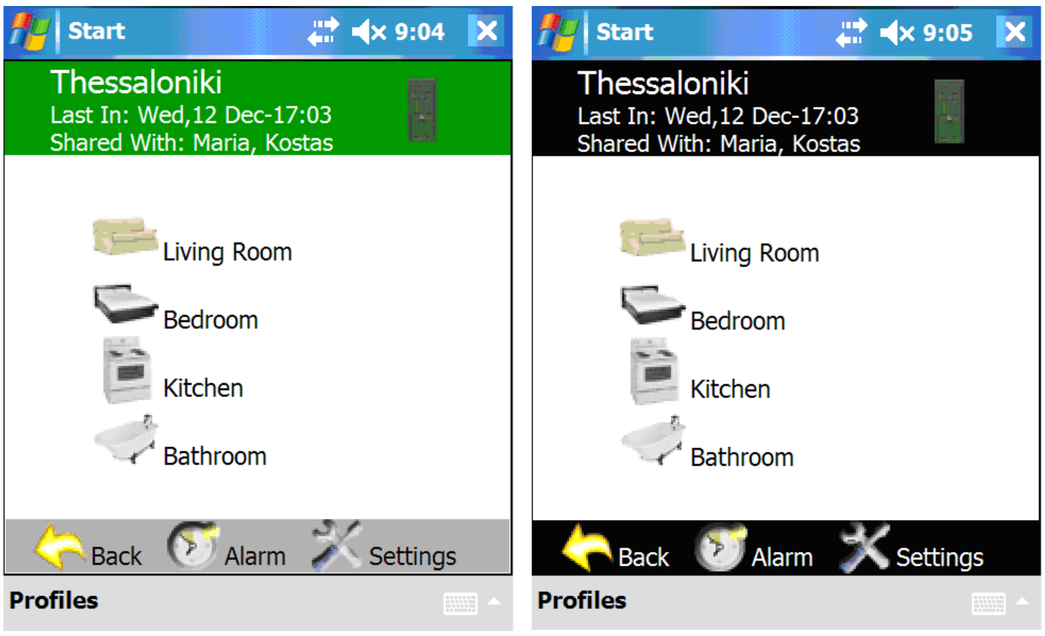
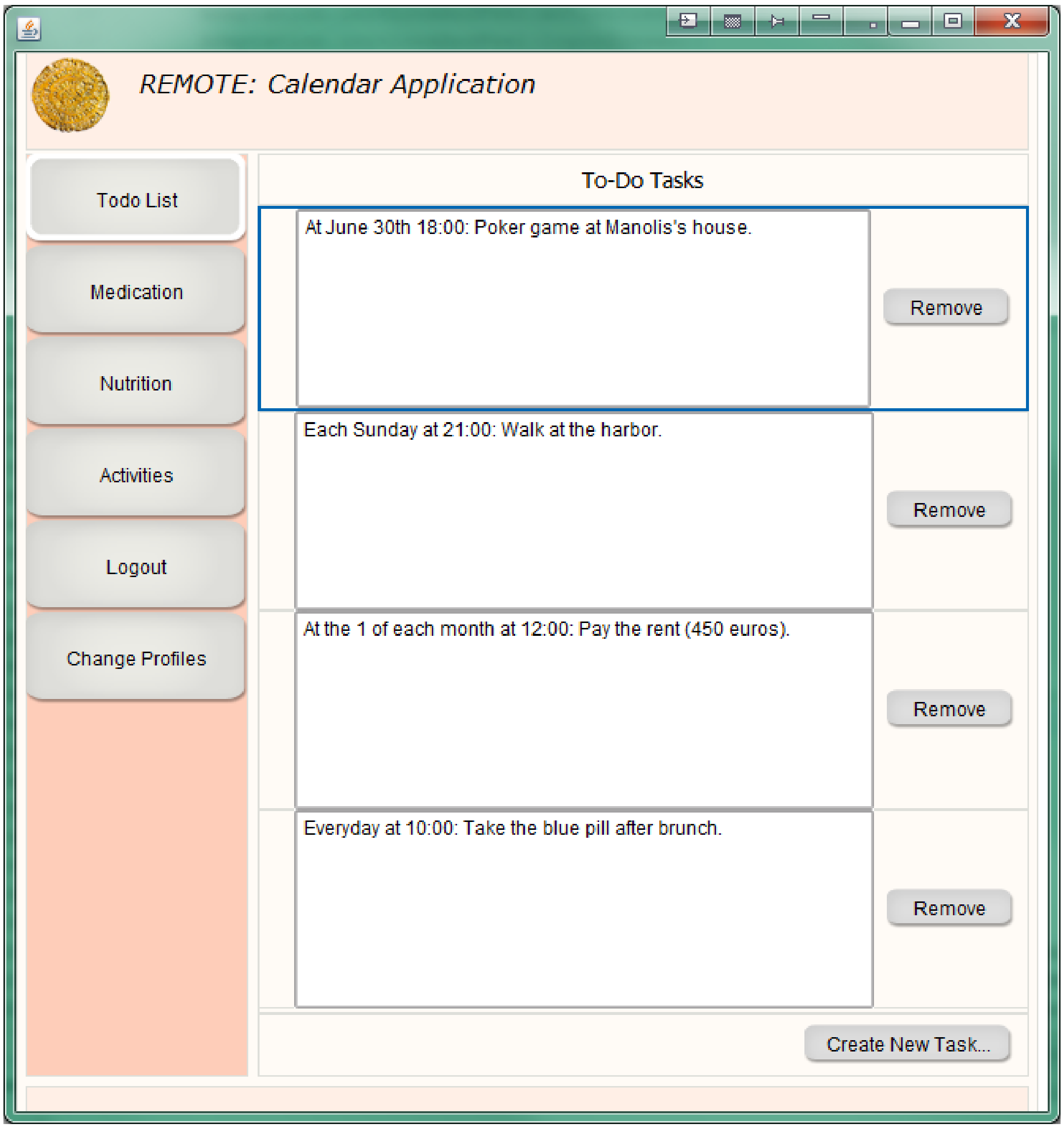


Figure 13. Example of widget adaptation in the ASK-IT Home Automation Application



At June 30th 18:00: Poker game at Manolis's house.

Each Sunday at 21:00: Walk at the harbor.

At the 1 of each month at 12:00: Pay the rent (450 euros).

Everyday at 10:00: Take the blue pill after brunch.

Figure 14. To-do-list in the REMOTE calendar application

Tables

Table 1. The design rationale of alternative image styles (from 0)

TASK: DISPLAY IMAGE				
Style:	**Image**	**As text**	**As link**	**Resizable thumbnail**
Targets:	-	Facilitate screen reader and low vision users in order not to be in difficulties with image viewing	Facilitate screen reader and low vision users in order not to be in difficulties with image viewing but with the capability to save or view an image	Viewing images in small size in order not to hold large size on the web page with the capability to enlarge the image to normal size when it is necessitated.
Parameters:	User (Default)	User (Blind or Low vision)	User (Blind or Low vision) and user preference)	User (preference)
Properties:	View image	Read image alternative text	Read image alternative text or and select linked named as the image alternative text to save or view the image	View image thumbnail and select it to view it in normal size
Relationships:	Exclusive	Exclusive	Exclusive	Exclusive

Table 2. Adaptation features of widgets in the Adaptable Widget Library

Buttons	Associated icon when idle, mouse over it, clicked or disabled Shortcut key Text Status (Enabled, Disabled) Tooltip's text and colors (foreground and background) Border Background and foreground color when clicked or idle Button's font Cursor appearance on mouse over i(e.g. hand cursor) Access key Vertical and horizontal text alignment Free space (gap) between button's icon and text
Check Box	Associated icon when enabled and checked, enabled and unchecked, disabled and checked or disabled and unchecked Shortcut key to check/uncheck the checkbox Text Status (Enabled, Disabled) Tooltip's text and colors (foreground and background) Border Background and foreground color Checkbox's font Cursor appearance on mouse over (e.g., hand cursor) Access Key Vertical and horizontal text alignment
Drop down menu	Background and foreground color of available and highlighted choices Choices' font
List	List orientation (vertical or horizontal) Background and foreground color of available and highlighted choices Choices' font Border around list component Tooltip text either one common for the list itself, or a different one for each choice Cursor appearance on mouse over (e.g. hand cursor) Access Key
Text Box	Text Maximum number of characters per line Text’s font Background color when this component is on or out of focus Border around text box Foreground color of the text when either enabled or disabled Cursor appearance when user hovers mouse over it (e.g. hand cursor) Access Key that facilitates traversal using keyboard (e.g. right arrow instead of Tab) Status (Enabled, Disabled) Editable status, whether the user can alter the contents of this text box Tooltip's text and colors (foreground and background) Highlight text color when selected by mouse or due to search facility
Password Text Box	Text Maximum number of characters per line Text’s font Background color when this component is on or out of focus Border around text box Foreground color of the text when either enabled or disabled Cursor appearance when user hovers mouse over it (e.g. hand cursor) Access Key that facilitates traversal using keyboard (e.g. right arrow instead of Tab) Status (Enabled, Disabled) Editable status, whether the user can alter the contents of this text box Tooltip's text and colors (foreground and background) Highlight text color when selected by mouse or due to search facility
Text Area	Text Maximum number of characters per line Text's Font Background (focused, not focused) Border around text box Foreground color of the text when either enabled or disabled Cursor appearance when user hovers mouse over it (e.g. hand cursor) Access Key that facilitates traversal using keyboard (e.g. right arrow instead of Tab) Status (Enabled, Disabled) Editable status, whether the user can alter the contents of this text box Tooltip's text and colors (foreground and background) Highlight text color when selected by mouse or due to search facility Maximum number of lines Type of text wrapping when text area is not wide enough
Radio Buttons	Text

	Background and foreground color when selected or not Border around radio button Radio button's text Font Cursor appearance when user hovers mouse over it (e.g. hand cursor) Status (Enabled, Disabled) Tooltip's text and colors (foreground and background) Foreground color when disabled Associated Icon when enabled and selected, enabled and unselected, disabled and selected or disabled and unselected Shortcut key to select the radio button Foreground and Background color when radio button is on or off focus
Hyperlink / Label	Text Foreground color when Enabled Background Color (inherited by parent container) Hyperlink's Font Tooltip's text and colors (foreground and background) Cursor appearance when user hovers mouse over it (e.g. hand cursor) Border around Hyperlink Status (Enabled, Disabled)
Table	Row height and margin between rows Foreground and Background color of currently selected cell Show grid (horizontal, vertical lines) Column width Border around Table Cells Background Color of Table Cell Tooltips' text and colors (foreground and background) Background and foreground color of the table Text's Font
Slider	Slider's Orientation (Horizontal or Vertical) Minimum and maximum value that user could select using slider Label for each discrete slider value (e.g. Start - End, 0 - 100 etc.) Visibility status of labels, major and minor ticks Background color of slider component Status (Enabled, Disabled) Border around slider component Tooltip's text and colors (foreground and background) Cursor appearance when user hovers mouse over it (e.g. hand cursor) Major and minor tick spacing (e.g. every fifth tick should be large - major-, while any other should be small -minor) Foreground color of ticks (little vertical lines below slider) Visibility status of the track Invert start with end (e.g. on vertical orientation start is the top of the slider while end is the bottom) Snap to ticks (limit user's selection only to ticks, e.g. when user slides cursor to 4.6, cursor should automatically be "attracted" to 5
Spinner	Background and Foreground Color Border around spinner Cursor appearance when user hovers mouse over it (e.g. hand cursor) Status (Enabled, Disabled) Text's Font Tooltip's text and colors (foreground and background)
Menu bar	Background color
Menu	Background color (inherited from menu bar) Foreground color Text Border around menu Associated icon when menu is opened and closed or enabled and disabled
Menu Item	Background and foreground color of each menu item Associated con when component is enabled or disabled, or when user hover its mouse over it
Progress Bar	Background and Foreground Color Progress's text font Border around bar Status (Enabled, Disabled) Tooltip's text and colors (foreground and background) Minimum and maximum value of the bar Progress bar's orientation (horizontal or vertical) Progress's text visibility status
Tooltips	Background and foreground color Text Text's Font Border around tooltip Cursor appearance when user hovers mouse over it (e.g. hand cursor) Status (Enabled, Disabled)
Tabs	Tab Layout Policy Tab placement

	Border around tab Tab's label Font Cursor appearance on mouse over (e.g., hand cursor) Mnemonic (visible and functional per tab) Status (Enabled, Disabled) either for all tabs or for a specific one Tooltip's text and colors (foreground and background) Associated icon with each tab when enabled or disabled, or when selected or unselected Tab's color when selected or not