



Available at  
[www.ElsevierComputerScience.com](http://www.ElsevierComputerScience.com)

POWERED BY SCIENCE @ DIRECT®

Interacting with Computers 16 (2004) 243–270

[www.elsevier.com/locate/intcom](http://www.elsevier.com/locate/intcom)

**Interacting  
with  
Computers**

# Unified user interface design: designing universally accessible interactions

Anthony Savidis<sup>a</sup>, Constantine Stephanidis<sup>a,b,\*</sup>

<sup>a</sup>*Institute of Computer Science, Foundation for Research and Technology—Hellas,  
Science and Technology Park of Crete, Heraklion, Crete, GR-71110, Greece*

<sup>b</sup>*Department of Computer Science, University of Crete, Greece*

Received 7 January 2002; revised 9 December 2002; accepted 14 December 2003

---

## Abstract

Designing universally accessible user interfaces means designing for diversity in end-users and contexts of use, and implies making alternative design decisions, at various levels of the interaction design, inherently leading to diversity in the final design outcomes. Towards this end, a design method leading to the construction of a single interface design instance is inappropriate, as it cannot accommodate for diversity of the resulting dialogue artifacts. Therefore, there is a need for a systematic process in which alternative design decisions for different design parameters may be supported. The outcome of such a design process realizes a design space populated with appropriate designed dialogue patterns, along with their associated design parameters (e.g. user- and usage-context-attribute values). This paper discusses the *Unified Interface Design Method*, a process-oriented design method enabling the organization of diversity-based design decisions around a single hierarchical structure, and encompassing a variety of techniques such as task analysis, abstract design, design polymorphism and design rationale.

© 2004 Elsevier B.V. All rights reserved.

**Keywords:** Dialogue design; Polymorphic task analysis; Design rationale; Interface adaptation; Unified user interfaces

---

---

\* Corresponding author. Address: Institute of Computer Science, Foundation for Research and Technology—Hellas, Science and Technology Park of Crete, Heraklion, P.O. Box 1385, Crete GR-71110, Greece. Tel.: +30-2810-391741; fax: +30-2810-391740.

E-mail address: [cs@ics.forth.gr](mailto:cs@ics.forth.gr) (C. Stephanidis).

## 1. Introduction

New User Interface design methods become necessary when there is a need to capture particular properties of interactive systems, which cannot be explicitly or sufficiently represented through existing design approaches. For instance, the identification of graphical constraints among interface objects is not usually carried out in the context of task analysis. Hence, if the explicit representation of graphical constraints is necessary (e.g. as an input to the implementation phase), a dedicated design process needs to be carried out for the extraction of such necessary design information. Also, in situations where a single design approach does not fulfill the requirements of the design process, the combination of alternative design techniques is applied, leading to hybrid design methodologies (e.g. combination of task analysis and graphic design methods).

New engineering methodologies and development tools emerge in cases where existing methods and instruments are not sufficient to effectively address the development challenges for new categories of interfaces. In the context of universal access, the need for revisiting interface engineering methodologies and tools has been already identified. While various approaches and tools exist today, they mainly fall within the domain of automatically adapted interfaces. Recent engineering work in the field of universal access has identified the genuine need for design spaces (Stephanidis and Savidis, 2001) that can support evolution and extensibility, while marrying together alternative design artifacts even for the same target design context; such design spaces may constitute the ground for advanced interface implementation processes. In this paper, we report a design methodology that reflects the overall objective of universal access, and in particular, the construction of evolving unified user interface design spaces. Interface engineering requirements are appropriately taken into consideration, without undue bias or restrictions by particular properties and implementation tactics of any specific engineering strategy.

### 1.1. The design problem

Universal design addresses potentially all users and usage-contexts—*anyone, anyplace, anytime*. Its main objective is to ensure that each end-user is given the most appropriate interactive experience, supporting accessible and high-quality interaction. Clearly, producing and enumerating distinct interface designs through the conduct of multiple design processes is an impractical solution, since the overall cost for managing in parallel such a large number of independent design processes, as well as for transforming each produced interface version to a target software implementation, would be unacceptable both for the design and the software implementation phases.

Instead, a design process is required which may lead to a single design outcome that appropriately links and organizes the differentiating aspects of the resulting interactive application, around common abstract design structures and patterns, making it far easier to: (a) map to a target software system implementation and (b) maintain, update and extend the design itself.

The need for introducing alternative design artifacts, even for the same specific design context (such as a particular sub-task), emerges from the fact that, in universal design, the design problem encompasses largely varying parameters, i.e. user- and

usage-context-parameters. Consequently, when designing for any particular dialogue design context, it is likely that differentiating values of those problem parameters dictate the design of diverse dialogue artifacts. This issue, introduces two important requirements for pursuing a suitable design method.

The first is that such a method should offer the capability to associate multiple alternative dialogue artifacts to a particular single design context, due to the varying design problem parameters, by enabling the unambiguous association of each alternative artifact with its corresponding values of the problem parameters.

The second is that the method should emphasize capturing of the more abstract structures and patterns inherent in the interface design, enabling the hierarchical incremental specialization towards the lower physical-level of interaction, making it possible to introduce alternative dialogue patterns as close to the physical design as possible. This will make it far easier for the design space to be updated and evolve, since modifications and extensions due to the consideration of additional values of the problem parameters (e.g. considering new user- and usage-context-attribute values) can be applied locally closer to the lower-levels of the design, without affecting the rest of the design space.

To demonstrate briefly the need for supporting alternative dialogue artifacts for the same design context, an example from a real-life application will be used. The AVANTI Web-browser (Stephanidis et al., 2001a) has been developed to enable Web-access by supporting adaptation to the particular user as well as to the context of use. During the interface design phase, while concentrating on the design context of the link dialogue task, alternative designs have been dictated, due to varying user- and usage-context-parameters considered, as shown in Fig. 1.

Since the differing artifacts have been part of the final AVANTI-browser interface design, the design representation formalism needed to enable their co-presence within the resulting design space, by clearly associating each artifact to the link-selection task and its corresponding values of the user- and usage-context-parameters. A loose design notation is employed in Fig. 1, to show hierarchical task analysis (sub-task sequencing is omitted for clarity), as well as the need for alternative incarnations of a single task (e.g. styles S2/S3 for link selection, styles S1/Se for load confirmation, and styles S4/S5 for link targeting). Following this example taken from the AVANTI browser, during run-time, depending on the particular end-user- and usage-context-attribute values, the appropriate corresponding implemented artifacts will only have to be activated.

### *1.2. The proposed solution*

The unified interface design method proposes a specific design process to cater for the management of an evolving design space, in which alternative design artifacts can be associated to variations of the design problem parameters. In this context, the conduct of the method, as well as the proposed design space representation will be discussed in detail, while a review of existing design methods will be firstly carried out, identifying the weaknesses of those methods to accommodate for diversity of the design-problem parameters. The key elements of the unified user interface design process are

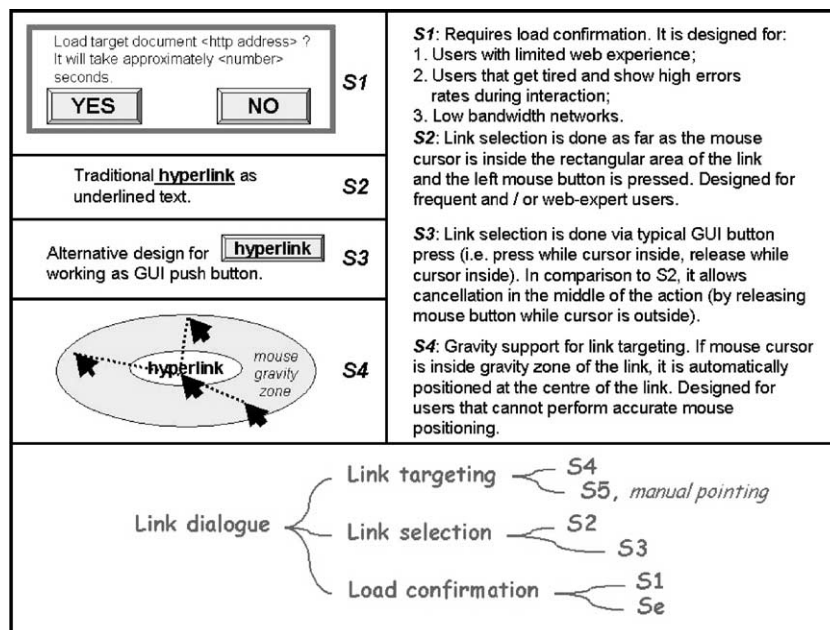


Fig. 1. Designing alternative artifacts for the *Link dialogue* task. *Se* is used to indicate 'empty' (i.e. no load confirmation dialogue supported). *S5* is the typical manual link targeting GUI dialogue through the mouse.

- Hierarchical design discipline building upon the notion of task analysis, empowered by the introduction of *task-level polymorphism*.
- Iterative design process model, emphasising *abstract task analysis* with incremental polymorphic physical specialization.
- Formalization of *run-time relationships* among the alternative design artifacts, when associated to the same design context.
- Documentation forms recording the *consolidated design rationale* of each alternative design artifact.

## 2. Related work

As it has been previously mentioned, in the context of universal design, given any particular design context within the overall design space, it is critical to enable co-presence of multiple alternative design artifacts, each associated with well-defined values of the design parameters. Hence, the designer should be enabled to organize those alternative artifacts together, as part of the final interface design, by specifying their association, both to the particular design context (e.g. a sub-task), and to the concrete set of values of the problem parameters (i.e. user- and usage-context-attribute values). This constitutes the key required ingredient of any design process that aims to support design for diversity, in the context of universal design. Additionally, the design process should provide all the necessary formal instruments for documenting the design rationale, as well as

the particular run-time relationships of the various alternative artifacts, for effective design and run-time organization.

A wide spectrum of key existing design methods are reviewed, their focus ranging from the low-level aspects of interaction to higher-level design artifacts, while addressing their appropriateness in accommodating for design diversity.

### *2.1. Task-oriented design methods*

Methods such as hierarchical task decomposition (Johnson et al., 1988), task-action grammars (Payne, 1984), and task-based design (Wilson and Johnson, 1995) are perhaps the most representative examples of the task-based design methods. Those techniques do not encompass the capability to differentiate and represent design alternatives for the same task, mapping to varying design problem parameters. Instead, the design outcome reflects only a specific instance of the design parameters, and conveys only a single task-based structure. Consequently, following the design formalism of task-based methods, the differentiation at various levels of the task model would need to be represented via distinct alternative task-hierarchies, i.e. alternative designs. Each of those distinct design instances would need to be potentially produced for each different snapshot of values of the design parameters, leading to a large number of alternative designs.

Clearly, such an approach becomes impractical both for the design as well as the implementation process, because of the prohibitive cost of maintaining and engineering a large set of alternative design instances. One might observe that all plausible task design instances are likely to exhibit structural similarities; hence, it would be expected that appropriate design and implementation procedures could take advantage of such commonalities by deploying re-usability techniques, while organizing appropriately the varying artifacts into different levels of abstraction. However, following the definition of those methods, it is clear that there is no way to support task-structure variability and explicit association of differentiated task-patterns with varying design parameters. Moreover, there is no additional support to designers for managing abstract design, other than the inherent modeling capability offered by the hierarchical task model.

### *2.2. Asynchronous models of user actions*

Such techniques are based on the asynchronous nature of user actions, providing models for representing concurrent user-activities, mainly through operators for expressing ordering relationships and progress control. They usually rely upon the theory of reactive systems, while the most representative examples in this category are CSP (Hoare, 1978), UAN (Hartson et al., 1990) and PPS (Olsen, 1990). These models suffer from two main problems: (a) they are mostly suited for representing temporal action-oriented dialogue properties, rather than constructional (i.e. they cannot represent structural interface relationships) and (b) they are quite low-level, by being very close to directly computable models, such as an implementation language, rather than to a design language, rendering them more appropriate for small design projects. Additionally, the various operators supplied for combining user actions correspond to run-time relationships (e.g. ‘this’ should be done before ‘that’), while they do not provide means for representing

design-oriented relationships (e.g. this sub-dialogue is defined for ‘those’ user attribute values).

### 2.3. *Interaction-object oriented design methods*

Such methods, which are less commonly used, emphasize the modeling of the constructional aspects of the User Interface, aiming mainly towards a design model which can be mapped automatically/semi-automatically, via appropriate transformation tools (such as special purpose compilers), to an implementation form. Design information such as task hierarchies may be defined as additional features within the object model. A relatively recent example of such a design technique is OBSM (Kneer and Szwillus, 1995), while earlier works such as Manheimer et al. (1990) and Desoi et al. (1989) have been mainly attributed to specific User Interface development tools, in which the object-based design specification is used to automatically generate the software implementation. Object-based methods are targeted towards the modeling of the structural interface properties, leaving no space for either behavioral or structural differentiation when the design parameters dictate diverse design decisions.

### 2.4. *Visual/graphical design methods*

Such methods are a type of straightforward physical design, in the sense that concrete visual interface snapshots are produced, via an iterative *sketching and refinement* interface prototyping process. The design process is conducted by managing design guidelines, usability criteria, artistic design, and task-oriented knowledge, towards more effective visual communication. In the context of this paper, such methods are considered to realize an *artifact generation* process, which can be employed at various stages of the interface design process, whenever the design needs to be instantiated into specific physical (e.g. visual) forms. Clearly, this type of technique cannot support the overall organization of the design process, starting from the early stages, when diverse design parameters impose different design decisions, but it is mostly suited for later stages when design artifacts need to be crystallized and finalized.

### 2.5. *Scenario-based design methods*

Such methods capture key usage-scenarios, annotated with design documentation which can be directly communicated to various levels of the design process, for further reviewing and enhancements. Such design documentation constitutes valuable reference material for early prototyping and engineering. Examples of use of scenario-based design are reported in Royer (1995) and Potts (1995). The employment of scenarios is mainly a complementary design technique, and has not been applied on its own to: (a) manage the conduct of the overall design process and (b) to facilitate the construction of the finalized design space as an appropriate organization of documented dialogue patterns.

## 2.6. Design rationale methods

Design rationale methods support argumentation about design alternatives and record the various design suggestions as well as their associated assessments. They are employed for the generation of design spaces, which capture and integrate design information (Belloti, 1993) from multiple sources, such as design discussions and diverse kinds of theoretical analyses. Questions, options and criteria (QOC) (McLean and McKerlie, 1995) design rationale is the most common method for constructing design spaces, capturing argumentation about multiple possible solutions to design problems. QOC can be used to characterize the nature of contributions from diverse approaches, highlighting strengths and weaknesses (Belloti, 1993).

It is clear that the employment of methods based on design rationale for constructing design spaces leads to a comprehensive collection of alternative solutions (i.e. candidate design decisions), addressing their respective design problems, from which the most appropriate for their purpose will be finally chosen (i.e. final design decisions). Hence, possible alternatives are likely to exist during the design process, but are removed from the final design, since only the best choice (for each particular design problem) should be documented as an input to the implementation phase. In comparison, this is fundamentally different to universal design, where alternative dialogue artifacts, representing final design decisions associated to particular user- and usage-context-attribute values, need to be concurrently present in the final design space.

## 2.7. Hybrid design methods

Hybrid design methods borrow elements mainly from task-based-, scenario-based-, and object-based-design techniques. Usually, one of these basic techniques becomes the centrepiece of the design process, while additional information can be extracted by performing parallel design steps through the deployment of any other method. For example, TADEUS (Stary, 1996) builds upon the task-model, providing also organizational and work-flow information. The technique proposed in Kandle (1995) is based on a task-model augmented with semi-formalized usage scenarios. In Butler et al. (1997), a mixture of task analysis, process simulation and object definition methods is proposed, to conduct a multi-level design process, in which initial work-flow design information is transformed into business object component (BOC) definitions, which are then converted to the object modelling technique (OMT), being mostly appropriate for the software design phase. Hybrid models, until now, have emphasized the quick transition to an implementation phase, and have been applied to design projects in which work-flow information is a necessary ingredient (i.e. business process engineering/re-engineering). The known methods do not demonstrate the capability to model diversity of design parameters, and inherent polymorphism of resulting artifacts, since the design outcomes can only reflect a single snapshot of values of the design parameters.

### 3. Fundamentals of unified user interface design

On the grounds of the above, in the context of a universal design process aiming to address multiple values of user- and usage-context-parameters, the *Unified User Interface Design Method* has been defined so as to address three main objectives:

- I. To enable the collection and effective organization of all potentially distinct design alternatives, as they are dictated for their particularly associated dialogue design contexts, into a single unified user interface design space.
- II. To produce a design space in which, for each alternative design artifact, its particular recorded design rationale, the run-time relationships with the rest of alternative artifacts within the same design context, as well as the specific associated problem parameters values, are represented in a well-defined computable form that can be directly translated by user interface developers into an implementation structure.
- III. To support design evolution, by enabling the effective extension of alternative dialogue artifacts at different design contexts, as new user- and usage-context-attribute values need to be addressed. While universal design is practically considered to be an ideal target, when intended to be addressed in an ‘one shot’ fashion during development, it turns to be an arguably manageable task when tackled through development strategies that support extension and evolution.

Some of the distinctive properties of this method are elaborated below by addressing its links with HCI design and by providing an overview of the outcomes and design deliverables in the context of a unified user interface design process.

#### 3.1. General characteristics

The Unified User Interface design method extends the traditional design inquiry by focusing explicitly on *polymorphism* as an aid to designing and populating the final design space with alternative required dialogue artifacts, when addressing diverse user- and usage- contexts. In this context, design polymorphism provably becomes the vehicle which empowers the design method with the ability to represent in the final design space, and in a well-organized way, the design pluralism that is inherent in any type of design process which aims to address a space of diverse problem-parameter values.

In terms of conduct, the method is related to hierarchical task analysis, with the distinction that alternative polymorphic decomposition schemes can be employed (at any point of the hierarchical task analysis process), where each decomposition seeks to address different values of the driving design parameters. This approach leads to the notion of *polymorphic task decomposition*, which is characterized by the pluralism of plausible design options consolidated in the resulting task hierarchy. Overall, the method is based on the new concept of polymorphic task decomposition, through which any task (or sub-task) may be decomposed in an arbitrary number of alternative sub-hierarchies. The design process realizes an exhaustive hierarchical decomposition of well-defined task categories, starting from the abstract level, by incrementally specializing in a polymorphic fashion (since different design alternatives are likely to be associated with



varying user- and usage-context-attribute values), towards the physical level of interaction. The outcomes of the method include: (a) the design space which is populated by collecting and enumerating design alternatives; (b) the polymorphic task hierarchy which comprises alternative concrete artifacts; and (c) for each produced design artifact, the recorded design rationale which has led to its introduction, together with run-time relationships with the rest of design alternative for the same sub-task.

### 3.2. *The design space*

Design alternatives are necessitated by the different design parameters and they may be attached to various levels of the overall task hierarchy, offering a rich insight into how a particular task/sub-task may be accomplished by different users in different contexts of use. Since users differ with regard to their abilities, skills, requirements and preferences, tentative designs should aim to accommodate the broadest possible range of capabilities across different contexts of use. Thus, instead of restricting the design activity to producing a single outcome, designers should strive to compile design spaces containing plausible alternatives.

### 3.3. *Polymorphic task hierarchies*

A polymorphic task hierarchy combines three fundamental properties: (a) *hierarchical decomposition*; (b) *polymorphism*; and (c) *task operators*. The hierarchical decomposition adopts the original properties of hierarchical task analysis (Johnson et al., 1988) for incremental decomposition of user tasks to lower level actions. The polymorphism property provides the design differentiation capability at any level of the task hierarchy, according to particular user- and usage-context-attribute values. Finally, task operators, which are based on the powerful communicating sequential processes (CSP) language for describing the behavior of the reactive systems (Hoare, 1978), enable the expression of dialogue control flow formulas for task accomplishment. Those specific operators, taken from the domain of reactive systems and process synchronization, have been selected due to their appropriateness in expressing temporal relationships of user actions and tasks (Fig. 2). However, designers may freely employ additional operators as needed (i.e. the set is not closed), or may choose to document dialogue sequencing and control outside the task-structure in natural language, when it engages more comprehensive algorithmic logic (e.g. consider the verbally documented precondition “if the logged user is a guest, no sign-in is required, else, the access privileges should be checked and the sign-in dialogue is activated before chat”).

The concept of polymorphic task hierarchies is shown in Fig. 3. Each alternative task decomposition is called a decomposition style, or simply a *style*, and is to be given by designers an appropriate descriptive name. Alternative task sub-hierarchies are attached to their respective styles. The example polymorphic task hierarchy of Fig. 3 indicates the way two alternative dialogue styles for a ‘delete file’ task can be designed, one exhibiting direct manipulation properties with object–function syntax (i.e. the file object is selected prior to operation to be applied) with no confirmation, the other realizing modal dialogue with




Operator	Explanation	Representation
<b>before</b>	Task sequencing, documenting that task A must be performed before task B.	
<b>or</b>	Task parallelism, documenting that task A may be performed before, after or in parallel to task B.	
<b>xor</b>	Task exclusive completion, documenting that either A or B must be performed, but not both.	
<b>*</b>	Task simple repetition, documenting that A may be performed zero or more times.	$A^*$
<b>+</b>	Task absolute repetition, documenting that A must be performed at least one time.	$A^+$

Fig. 2. Basic task operators in the Unified User Interface design method.

a function–object syntax (i.e. the delete function is selected, followed by the identification of the target file) and confirmation.

Additionally, the example demonstrates the case of physical specialization. Since ‘selection’ is an abstract task, it is possible to design alternative ways for physically

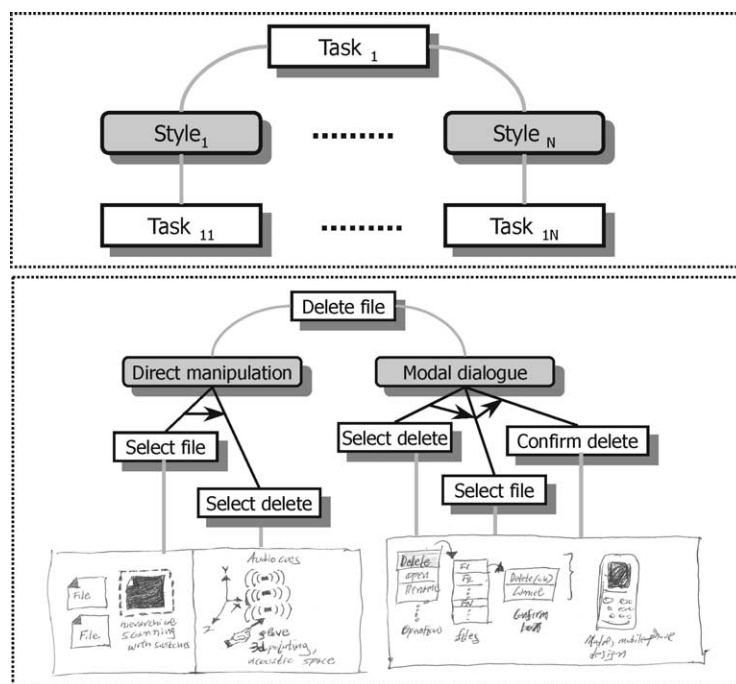


Fig. 3. The polymorphic task hierarchy concept, where alternative decomposition ‘styles’ are supported (upper part), and an exemplary polymorphic decomposition, which includes physical design annotation (lower part).

instantiating the selection dialogue (see Fig. 3, lower-part): via scanning techniques for motor-impaired users, 3D hand-pointing on 3D-auditory cues for blind people, enclosing areas (e.g. irregular ‘rubber banding’) for sighted users, and Braille output and keyboard input for deaf–blind users. The Unified User Interface design method does not require the designer to follow the polymorphic task decomposition all the way down the user-task hierarchy, until primitive actions are met. A non-polymorphic task can be specialized at any level, following any design method chosen by the interface designer. For instance, in Fig. 3 (lower part) graphical mock-ups are employed to describe each of the alternative physical instantiations of the abstract selection task. It should be noted that the interface designer is not constrained to using a particular model, such as CSP operators, for describing user actions for device-level interaction (e.g. drawing, drag-and-drop, concurrent input). Instead, an alternative may be preferred, such as an event-based representation, e.g. ERL (Hill, 1986) or UAN (Hartson and Hix, 1989).

As discussed in detail in Sections 3.4–4.4, design polymorphism entails a decision-making capability for context-sensitive selection among alternative artifacts, so as to assemble a suitable interface instance, while task operators support temporal relationships and access restrictions applied to the interactive facilities of a particular interface instance.

### 3.4. Adaptation-oriented design rationale

When a particular task is subject to polymorphism, alternative sub-hierarchies are designed, each being associated to different user- and usage-context-parameter values. A running interface, implementing such alternative artifacts, should encompass decision-making capability, so that, before initiating interaction with a particular end-user, the most appropriate of those artifacts are activated for all polymorphic tasks. Hence, polymorphism can be seen as a technique potentially increasing the number of alternative interface instances represented by a typical hierarchical task model. In this sense, the polymorphic task model theoretically represents the power set of all plausible design instantiations.

If polymorphism is not applied, a task model represents a singular interface design (i.e. a design instance), on which further run-time adaptation is restricted; in other words, *there is a fundamental link between adaptation capability and polymorphism on design artifacts*. This issue will be further clarified with the use of an example. Consider the case where the design process reveals the necessity of having multiple alternative sub-dialogues available concurrently to the user for performing a particular task. This scenario is related to the notion of multimodality, which can be more specifically called *task-level multimodality*, in analogy to the notion of *multimodal input*, which emphasizes pluralism at the input-device level. The example will consider the physical design artifact of Fig. 4, which depicts two alternative dialogue patterns for file management: one providing direct manipulation facilities, and another employing command-based dialogue. Both artifacts can be represented as a part of the task-based design, in two ways:

- (a) Through polymorphic decomposition (see Fig. 5, upper part), where each of the two dialogue artifacts is defined as a distinct alternative style; the two resulting styles are defined as being *compatible*, which implies that they may co-exist at run-time

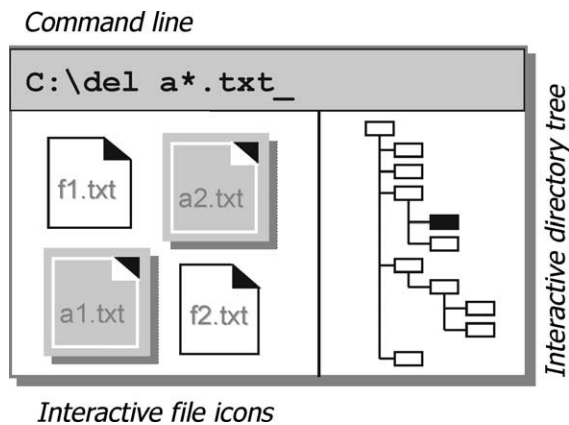


Fig. 4. A design scenario for alternative concurrent sub-dialogues, in order to perform a single task (i.e. task multimodality for file management).

(i.e. the end-user may freely use the command line or the interactive file manager interchangeably).

- (b) By unimorphic decomposition (see Fig. 5, lower part), where the two artifacts are defined to be concurrently available to the user, *within the same interface instance*, via the *or* operator; in this case, the interface design is ‘hard-coded’, representing a single interface instance, without needing further decision making.

The advantages of the polymorphic approach are (a) it is possible to make only one of the two artifacts available to the user, depending on user parameters; (b) even if, initially,

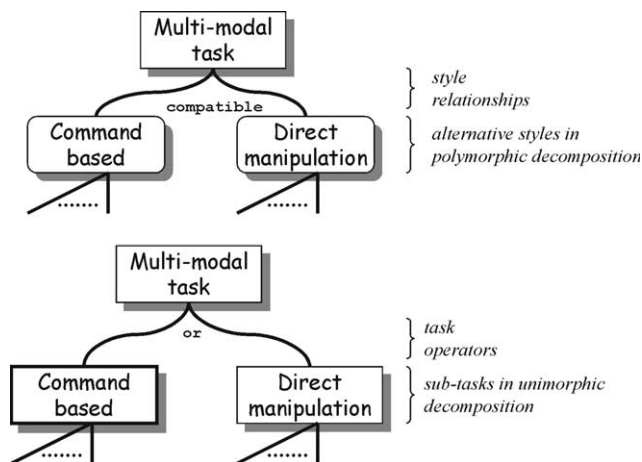


Fig. 5. Two ways for representing design alternatives when designing for task-level multimodality: (a) via polymorphism, adding run-time control on pattern activation (top version); and (b) via the *or* operator, hard-coding the two alternatives in a single task implementation (bottom version).

both artifacts are provided to end-users, when a particular preference is dynamically detected for one of those, the alternative artifact can be dynamically disabled; and (c) if more alternative artifacts are designed for the same task, the polymorphic design is directly extensible, while the decomposition-based design would need to be turned into a polymorphic one (except from the unlikely case where it is still desirable to provide all defined sub-dialogues concurrently to the user).

From the implementation point of view, the polymorphic decomposition requires the explicit representation of the associated design rationale in a directly computable form, such as logic expressions, preconditions, and decision algorithms. Additionally, particular emphasis is put in the translation of the design rationale to a run-time decision-making kernel for conditional dialogue component activation.

#### 4. The conduct of unified user interface design

This section will provide a consolidated account of how the unified interface design method can be practiced.

##### 4.1. Categories of polymorphic artifacts

In the unified user interface design method, there are three categories of design artifacts, all of which are subject to polymorphism on the basis of varying user- and usage-context-parameter values. These three categories are (Fig. 6):

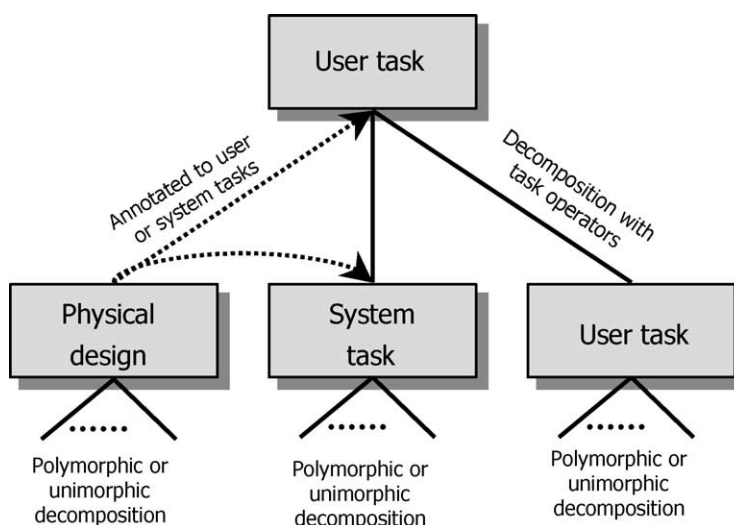


Fig. 6. The three artifact categories in the Unified User Interface design method, for which polymorphism may be applied, and how they relate to each other.

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

System tasks and user tasks may be freely combined within task ‘formulas’, defining how sequences of user-initiated actions and system-driven actions interrelate. The physical design, providing the interaction context, is always associated with a particular user or system task. It provides the physical dialogue pattern associated to a task-structure definition. Hence, it plays the role of annotating the task hierarchy with physical design information. An example of such annotation is shown in Fig. 2, where the physical designs for the select delete task are explicitly depicted.

In some cases, given a particular user task, there is a need for differentiated physical interaction contexts, depending on user- and usage-context-parameter values. Hence, even though the task decomposition is not affected (i.e. the same user actions are to be performed), the physical design may have to be altered. One such representative example is relevant to changing particular graphical attributes, based on ethnographic user attributes. For instance, Marcus (1996) discusses the choice of different iconic representations, background patterns, visual message structure, etc., on the basis of cultural background.

However, there are also cases in which the alternative physical designs are dictated due to alternative task structures (i.e. polymorphic tasks). In such situations, each alternative physical design is directly attached to its respective alternative *style* (i.e. sub-hierarchy).

In summary, the rule for identifying polymorphism for physical design artifacts is

- If alternative physical designs are assigned to the same task, then attach a polymorphic physical design artifact to this task; the various alternative designs depict the styles of this polymorphic physical artifact (see Fig. 7, part A).
- If alternative designs are needed due to alternative task structures (i.e. task-level polymorphism), then each alternative physical design should be attached to its respective style (see Fig. 7, part B).

#### 4.2. Design steps in polymorphic task decomposition

User tasks, and in certain cases, system tasks, need not always be related to physical interaction, but may represent *abstraction* on either user- or system- actions. For instance, if the user has to perform a selection task, then, clearly, the physical means of performing such a task are not explicitly defined, unless the dialogue steps to perform selection are further decomposed. This notion of continuous refinement and hierarchical analysis, starting from higher-level abstract artifacts, and incrementally specializing towards

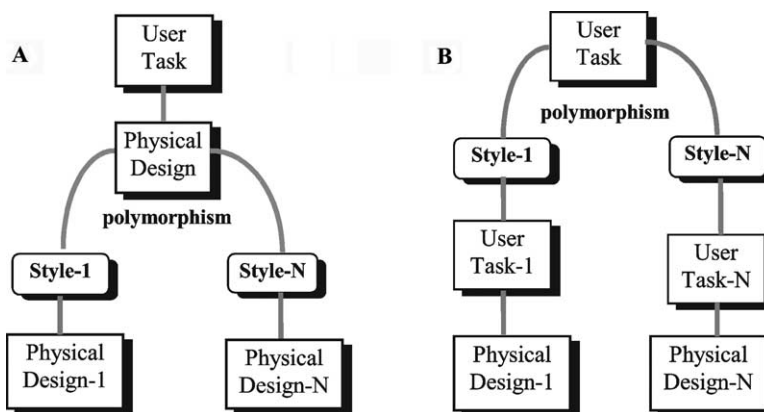


Fig. 7. Representation of alternative physical artifacts: (A) in the case of the same non-polymorphic task; and (B) in the case where polymorphic task decomposition is needed.

the physical level of interaction, is fundamental in the context of hierarchical behavior analysis, either regarding tasks that humans have to perform (Johnson et al., 1988), or when it concerns functional system design (Saldarini, 1989). At the core of the unified user interface design method is the *polymorphic* task decomposition process, which follows the methodology of abstract task definition and incremental specialization, where tasks may be hierarchically analyzed through various alternative schemes.

In such a recursive process, involving tasks ranging from the abstract task level, to specific physical actions, decomposition is applied either in a traditional *unimorphic* fashion, or by means of alternative styles. The overall process is shown in Fig. 8; the decomposition starts from abstract- or physical-task design, depending on whether

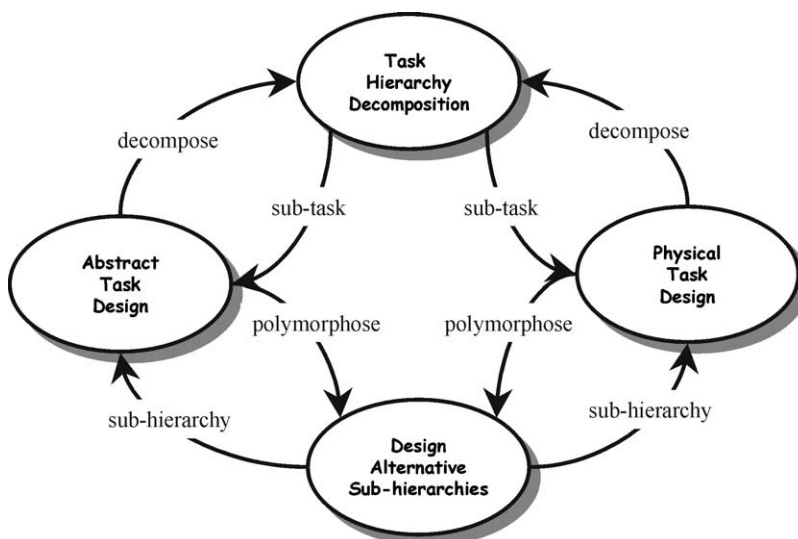


Fig. 8. The polymorphic task decomposition process in the Unified User Interface design method.

top-level user tasks can be defined as being abstract or not. Next, follows the description of the various transitions (i.e. design specialization steps), from each of the four states illustrated in the process state diagram of Fig. 8.

#### 4.2.1. Transitions from the ‘abstract task design’ state

An abstract task can be decomposed either in a polymorphic fashion, if user- and usage-context-attribute values pose the necessity for alternative dialogue patterns, or in a traditional manner, following a unimorphic decomposition scheme. In the case of a unimorphic decomposition scheme, the transition is realized by a *decomposition* action, leading to the *task-hierarchy decomposition* state. In the case of a polymorphic decomposition, the transition is realized by a *polymorphose* action, leading to the *alternative sub-hierarchies design* state.

#### 4.2.2. Transitions from the ‘alternative sub-hierarchies design’ state

Reaching this state means that the required alternative dialogue styles have been identified, each initiating a distinct sub-hierarchy decomposition process. Hence, each such sub-hierarchy initiates its own instance of polymorphic task decomposition process. While initiating each distinct process, the designer may either start from the *abstract task design* state, or from the *physical task design* state. The former is pursued if the top-level task of the particular sub-hierarchy is an abstract one. In contrast, the latter option is relevant in case that the top-level task explicitly engages physical interaction issues.

#### 4.2.3. Transitions from the ‘task-hierarchy decomposition’ state

From this state, the sub-tasks identified need to be further decomposed. For each sub-task at the abstract level, there is a *sub-task* transition to the *abstract task design* state. Otherwise, if the sub-task explicitly engages physical interaction means, a *sub-task* transition is taken to the *physical task design* state.

#### 4.2.4. Transitions from the ‘physical task design’ state

Physical tasks may be further decomposed either in a unimorphic fashion, or in a polymorphic fashion. These two alternative design possibilities are indicated by the *decompose* and *polymorphose* transitions, respectively.

### 4.3. Designing alternative styles

The polymorphic task model provides the design structure for organizing the various alternative dialogue patterns into a unified form. Such a hierarchical structure realizes the fusion of all potential distinct designs, which may be explicitly enumerated given the relevant design parameters. Apart from the polymorphic organization model, the following primary issues need to be also addressed: (i) when polymorphism should be applied; (ii) which are the user- and usage-context-attributes that need to be considered; (iii) which are the run-time relationships among alternative styles; and (iv) how the adaptation-oriented design rationale, connecting the designed styles with particular user- and usage-context-attribute values, is documented.



#### 4.3.1. Identifying levels of potential polymorphism

In the context of the unified user interface design method, and as part of the polymorphic task decomposition process, designers should always assert that every decomposition step (i.e. those realized either via the *polymorphose* or through the *decompose* transitions of Fig. 8) satisfies all constraints imposed by the combination of target user- and usage-context-attribute values. These two classes of parameters will be referred to collectively as *decision parameters/attributes*. When during the design process, there are situations where a particular task decomposition (for user- or system- tasks, as well as for physical design) does not address some combination(s) of the decision attribute values (i.e. those values are excluded), the design for this task is considered to be incomplete. Such an issue can be resolved during a later design iteration, by constructing the necessary alternative sub-hierarchy(-ies) for the subject task, appropriately addressing the excluded decision attribute values.

#### 4.3.2. Constructing the space of decision parameters

This section will discuss the definition of user attributes, which are of primary importance, while the construction of context attributes (e.g. environment and computing-platform parameters) may follow the same representation approach. In the unified user interface design method, end-user representations may be developed using any suitable formalism encapsulating user characteristics in terms of attribute-value pairs. There is no predefined set of attribute categories. Some examples of attribute classes are general computer-use expertise, domain-specific knowledge, role in an organizational context, motor abilities, sensory abilities, mental abilities, etc.

The value domains for each attribute class are chosen as a part of the design process (e.g. by interface designers, or Human Factors experts), while the value sets need not be finite. The broader the set of values, the higher the differentiation capability among individual end-users. The unified user interface design method does not pose any restrictions as to the attribute categories considered relevant, or the target value domains of such attributes. Instead, it seeks to *provide only the framework in which the role of user- and usage-context-attributes constitute an explicit part of the design process*. It is the responsibility of interface designers to choose appropriate attributes and corresponding value ranges, as well as to define appropriate design alternatives. A simple example of an individual user-profile, complying with the attribute/value scheme, is shown in Fig. 9. For simplicity, designers may choose to elicit only those attributes from which differentiated design decisions are likely to emerge.

Computer Knowledge	expert	frequent	average	casual	naive	
Web Knowledge	very good	good	average	some	limited	none
Ability to Use Left Hand	perfect	good	some	limited	none	
Age	30					
Gender	Male		Female			

Fig. 9. An example of a user-profile, as a collection of values from the value domains of user-attributes, extended from Stephanidis et al. (1997).

#### 4.3.3. Relationships among alternative styles

The need for alternative styles emerges during the design process, when it is identified that some particular user- and/or usage-context-attribute values are not addressed by the dialogue artifacts that have already been designed. Starting from this observation, one could argue that ‘all alternative styles, for a particular polymorphic artifact, are mutually exclusive to each other’ (in this context, exclusion means that, at run-time, only one of those styles may be ‘active’).

However, there exist cases in which it is meaningful to make artifacts belonging to alternative styles, concurrently available in a single adapted interface instance. For example, in Fig. 4 we have discussed how two alternative artifacts for file management tasks, a direct-manipulation one and a command-based one, can both be present at run-time. In the unified user interface design method, four design relationships between alternative styles are distinguished (Fig. 10), defining whether alternative styles may be concurrently present at run-time. These four fundamental relationships reflect pragmatic, real-world design scenarios.

**Exclusion.** The exclusion relationship is applied when the various alternative styles are deemed usable only within the space of their target user- and usage-context-attribute values. For instance, assume that the two alternative artifacts for a particular sub-task are being designed, aiming to address the ‘user expertise’ attribute: one targeted to users qualified as ‘novice’, and the other targeted to ‘expert’ users. Then, these two are defined to be mutually exclusive to each other, since it is probably meaningless to concurrently activate both dialogue patterns. For example, at run-time a novice user might be offered a functionally ‘simple’ alternative of a task, where an expert user would be provided with additional functionality and greater ‘freedom’ in selecting different ways to accomplish the same task.

**Compatibility.** Compatibility is useful among alternative styles for which the concurrent presence during interaction allows the user to perform certain actions in alternative ways, without introducing usability problems. The most important application of compatibility is for *task-multimodality*, as it has been previously discussed (Fig. 5, where the design artifact provides two alternative styles for interactive file management).

**Substitution.** Substitution has a very strong connection with adaptivity techniques. It is applied in cases where, during interaction, it is decided that some dialogue patterns need to be substituted by others. For instance, the ordering and the arrangement of certain operations may change based on monitoring data collected during interaction, through

<b>Exclusion</b>	<i>Relates many styles. Only one from the alternative styles may be present.</i>
<b>Compatibility</b>	<i>Relates many styles. Any of the alternative styles may be present.</i>
<b>Substitution</b>	<i>Relates two groups of styles together. When the second is made “active” at run-time, the first should be “deactivated”.</i>
<b>Augmentation</b>	<i>Relates one style with a group of styles. On the presence of any style from the group at run-time, the single style may be also “activated”.</i>

Fig. 10. Design relationships among alternative styles, and their run-time interpretation.

which information such as frequency of use and repeated usage patterns can be extracted. Hence, particular physical design styles would need to be ‘canceled’, while appropriate alternatives would need to be ‘activated’. This sequence of actions, i.e. ‘cancellation’ followed by ‘activation’, is the realization of substitution. Thus, in the general case, substitution involves two groups of styles: some styles are canceled and substituted by other styles that are activated afterwards.

*Augmentation.* Augmentation aims to enhance the interaction with a particular style that is found to be valid, but not sufficient to facilitate the user’s task. To illustrate this point, let assume that, during interaction, the user interface detects that the user is unable to perform a certain task. This would trigger an adaptation (in the form of adaptive action) aiming to provide task-sensitive guidance to the user. Such an action should not aim to invalidate the active style (by means of style substitution), but rather to augment the user’s capability to accomplish the task more effectively, by providing informative feedback. Such feedback can be realized through a separate, but compatible, style. It follows, therefore, that the augmentation relationship can be assigned to two styles when one can be used to enhance the interaction while the other is active. Thus, for instance, the adaptive prompting dialogue pattern, which provides task-oriented help, may be related via an augmentation relationship with all alternative styles (of a specific task), provided that it is compatible with them.

#### 4.4. Recording design documentation in polymorphic decomposition

During the polymorphic task decomposition process, there is a set of design parameters that need to be explicitly defined (i.e. given specific values) for each alternative sub-hierarchy defined. The aim is to capture, for each *sub-task*, the design logic for deciding possible alternative styles, by directly associating user-, usage-context-parameters and design goals with the constructed artifacts (i.e. styles). These parameters are: (i) *users and usage-contexts* (specific user- and usage-context- attribute values addressed by a style); (ii) *targets* (concrete design goals for a particular style); (iii) *properties* (the specific differentiating/distinctive interaction properties of a style, potentially in comparison to other styles); and (iv) *relationships* (how is a style related with other alternative styles). The values of these parameters are recorded during the decomposition process for each style in the form of a table. In Fig. 11, an example is shown for the definition of these parameters regarding the two alternative styles for delete file task (Fig. 3).

The purpose of design documentation is to capture design rationale associated to the various polymorphic artifacts. In this context, the notion of design rationale has

Task: Delete File	
<b>Style:</b> Direct Manipulation	<b>Style:</b> Modal Dialogue
<b>Users &amp; Contexts:</b> Expert, Frequent, Average	<b>Users &amp; Contexts:</b> Casual, Naïve.
<b>Targets:</b> Speed, naturalness, flexibility.	<b>Targets:</b> Safety, guided steps.
<b>Properties:</b> Object first, function next.	<b>Properties:</b> Function first, object next.
<b>Relationships:</b> Exclusion (with all).	<b>Relationships:</b> Exclusion (with all).

Fig. 11. An example of recording design documentation.

a fundamentally different objective with respect to well-known design space analysis methods. In the latter case, design rationale mainly represents argumentation about design alternatives and assessments (Bellotti, 1993) before reaching final design decisions, while in our case, design rationale records the different user- and usage-context-attributes, as well as design objectives underpinning the already made (i.e. final) design decisions.

The set of four parameters previously defined serves mostly as an ‘indexing’ method for organizing final design decisions, with primary keys the particular *sub-task*, and the *users and usage-contexts* parameters. The outcome of the unified user interface design approach is a single hierarchical artefact, composed of user- and usage-context-oriented final design decisions, associated to directly computable parameters (i.e. task, user- and usage-context-attributes). This unified structure is very close to an implementation-oriented software organization model, thus, potentially constituting a valuable asset for the implementation phase.

## 5. Scenarios of unified user interface design

This section will briefly discuss specific design scenarios occurred in the application of the unified interface design process in the development of the adaptable and adaptive AVANTI Web-browser (Stephanidis et al., 2001a).

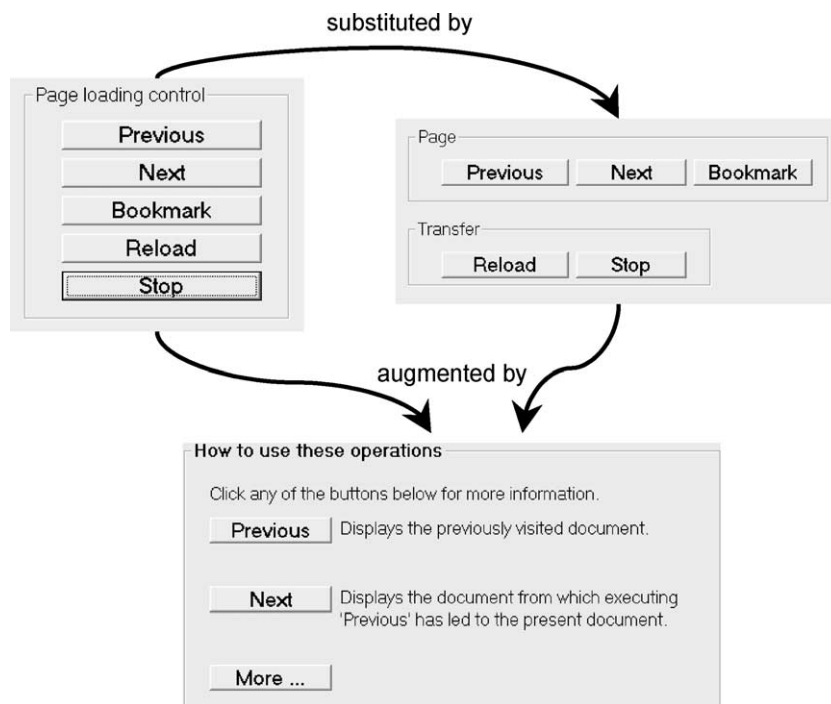
Some key design artifacts will be discussed, in situations where polymorphism has been applied, for the design of the ‘adaptable and adaptive’ Web browser. The target user audience for the AVANTI project has been able-bodied, motor-impaired and blind users, with differing computer-use expertise, supporting use in various physical environments (office, home, public terminals at stations/airports, PDAs, etc).

### 5.1. ‘Link dialogue’ task

In existing Web browsers (e.g. Netscape Communicator™, Microsoft Internet Explorer™ and SUN HotJava™), links in Web documents are activated by pressing the left mouse button while the cursor resides within the area of a link. In Fig. 1 earlier, the polymorphic design of the link dialogue (for textual links) has been presented. There are three steps in link selection, according to the new design: (a) targeting to the desirable link (can be done via two styles, S4 and S5); (b) actually selecting a link (which can be done in two alternative ways, S2 and S3); and (c) requiring confirmation for loading target document (also done in two ways, S1 and Se). The Se denotes an empty alternative sub-hierarchy (i.e. the task is considered to be directly accomplished without any user actions). The design documentation for polymorphism is provided in Fig. 12 (only a brief summary

<ul style="list-style-type: none"> <li>• S2 mutually exclusive with S3.</li> <li>• S1 mutually exclusive with Se.</li> <li>• S4 compatible with S5.</li> </ul>	<ul style="list-style-type: none"> <li>• S1 substitutes Se dynamically (if high error rates are detected, or in network bandwidth reduction).</li> </ul>
<ul style="list-style-type: none"> <li>• Se substitutes S1 dynamically (if in a satisfactory interaction history, link activation has been always followed by positive confirmation).</li> </ul>	

Fig. 12. Relationships among alternative styles of *Link Selection* task.

Fig. 13. Designed styles for the *Page Loading Control* task.

is presented for clarity), together with the relationships between the designed styles (notice that dynamic style updates need to be explicitly mentioned).

## 5.2. 'Document loading control' task

This task concerns typical operations that browsers provide to enable users control the Web page to be loaded and displayed (e.g. forward/backward, home, reload/stop, bookmarking, load options). In Fig. 13, two alternative style designs are shown, primarily

- |  |
|--|
| <ul style="list-style-type: none"> <li>• The top-left style is the style to be initially active ("casual" or "naïve" values on "application expertise" user attribute).</li> <li>• The top-right style is to dynamically substitute the previous style, in case that during monitoring it is observed that the user has used the operations successfully and became familiar with them. The new style groups logically operations with a title, and prepares the ground for the more advanced group called "options" to be included in future interaction sessions with this user.</li> <li>• The bottom style augments the particular active style, and it provides adaptive prompting / helping for carrying out the operations, in cases where inability to perform the task or high error rates are dynamically detected.</li> </ul> |
|--|

Fig. 14. Relationships between designed styles for *Page Loading Control* task.

designed for casual and naive users, which provide only a sub-set of the full range of operations (more advanced functions are only revealed to expert/frequent/average users). The relationships between the two styles are indicated in Fig. 14. Also, an adaptive prompting style is designed for the Page Loading Control task, aiming to help the user in performing this task, thus being an augmentation of the previous two alternative styles.

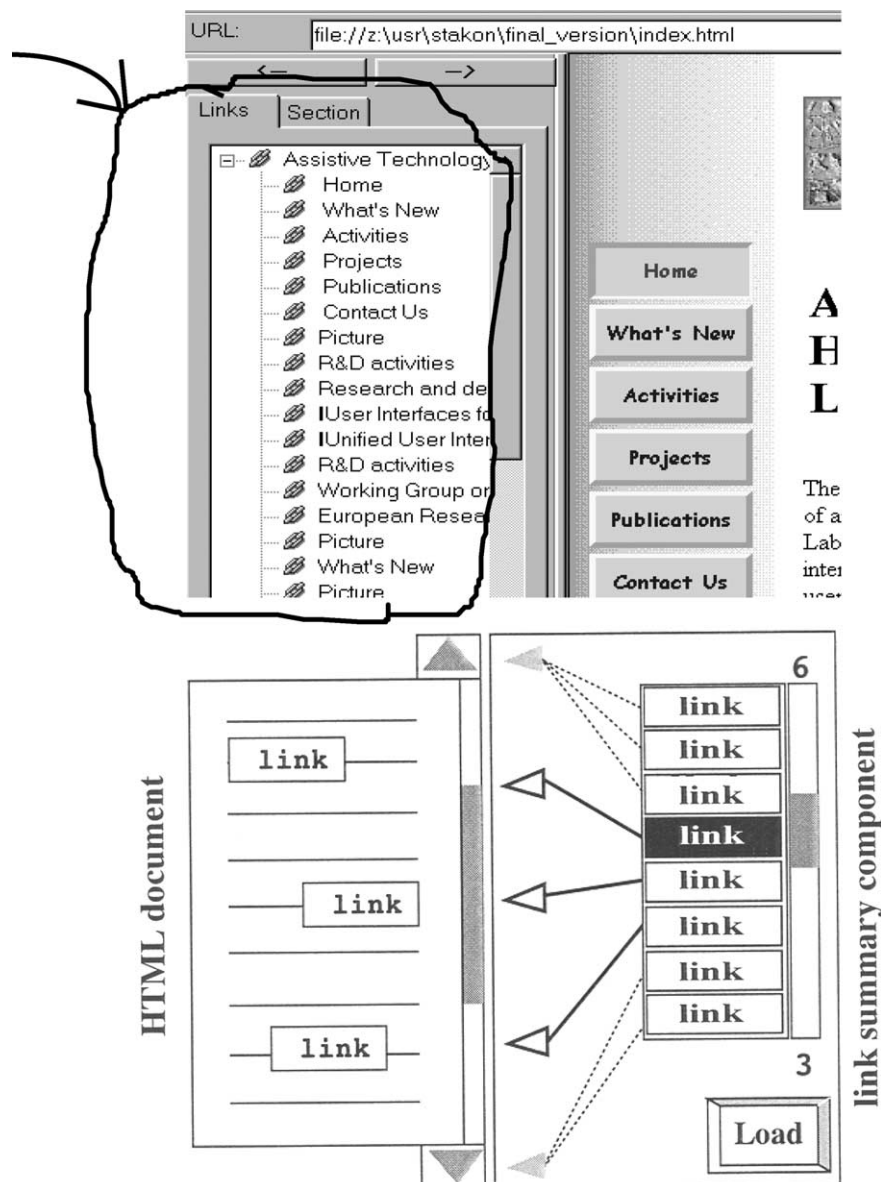


Fig. 15. The Link Summary styles: S1, (top) and S2 (bottom) for *Page Browsing* task.

### 5.3. 'Page Browsing' task

The requirement to provide accessibility of the resulting browsers by motor-impaired users, necessitated the design of dialogues for enabling motor-impaired users to explore

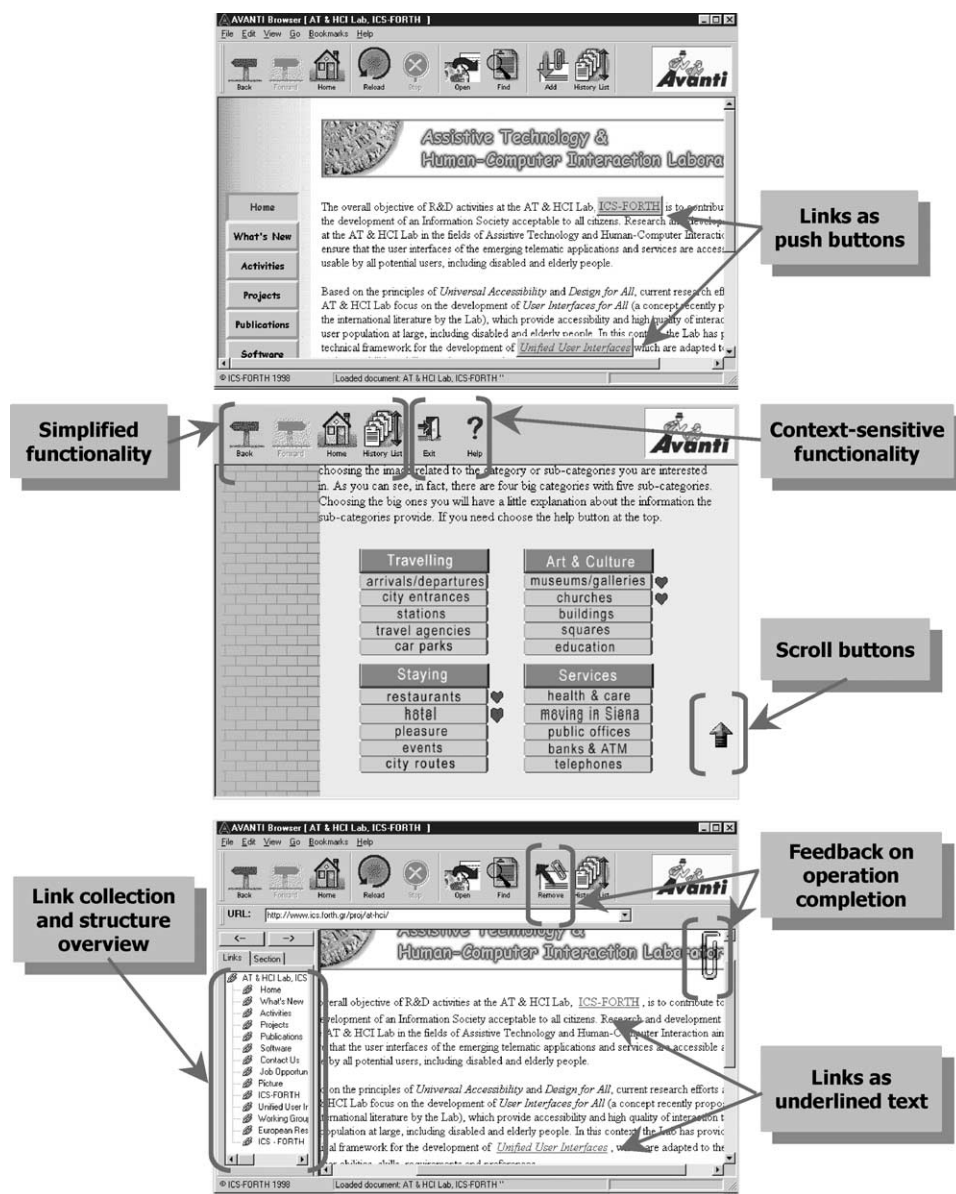


Fig. 16. Three different interface versions of the AVANTI browser produced by adaptation-oriented selection of different alternative styles.



page contents and activate links. One globally applied technique has been to support hierarchical scanning of all objects in the interface via binary switches. In this case, motor-impaired users can have access on the original visual interface designed for able users, through alternative switch-based input techniques. This approach proved to be very good for all tasks, except from the case of page browsing as users spend a lot of time for switching between the scroll-bar (of the page presentation) and the visible page contents in order to identify desirable information/links. This has resulted in two alternative styles, illustrated in Fig. 15, which augment the page presentation style, and are mutually exclusive. In Fig. 15, style S1, the summary of links is presented in a window on the left of the Web page (i.e. all links collected and presented together), while in style S2, the document display on the left is automatically adjusted, so that the highlighted link is always visible. Dashed arrows indicate that document context including the associated link is above or below visible portion, while solid arrows are attached at visible links and point to the exact link position in the document visible area; the number displayed above or below the scrollbar of the 'inks summary listbox' indicates how many other links are included above or below the first or last displayed link within this listbox.

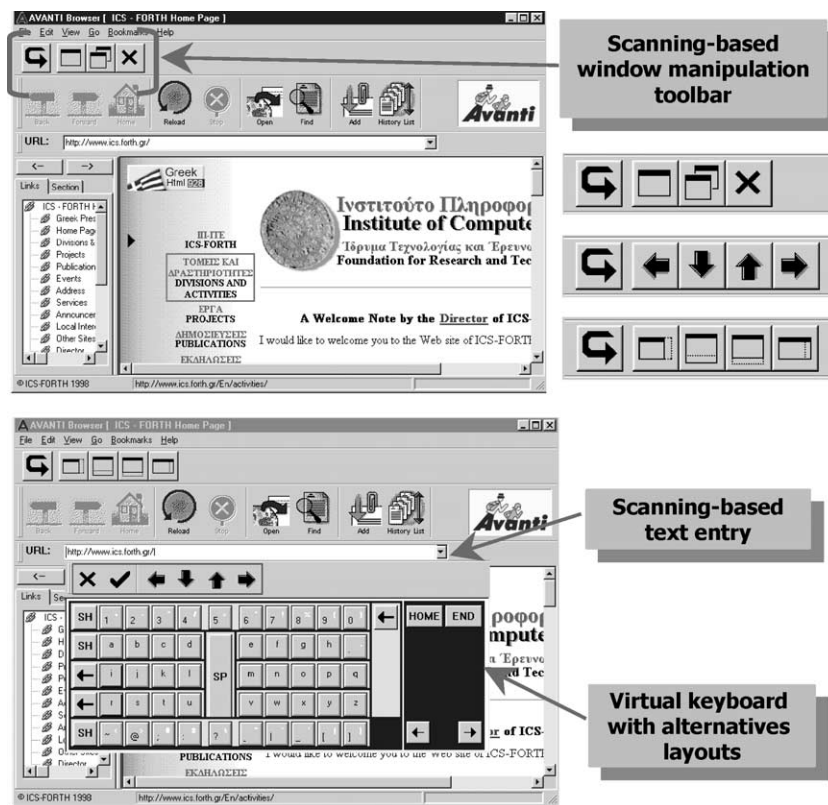


Fig. 17. Alternative augmentation-oriented styles for motor-impaired user access activated at run-time in the AVANTI browser.



In Fig. 16, the combination of alternative styles at run-time results in different automatically produced interface versions for the AVANTI browser. Each of the different styles corresponds to specific user- and usage-context-attribute values (for instance, the middle version, with toolbars and link-summary pane removed corresponds to kiosk installations). Finally, in Fig. 17, augmentation styles to provide accessible alternative interaction for motor-impaired user via switch-based scanning are activated, to enable accessible Web browsing. An extended discussion on the unified user interface design and implementation of the AVANTI Web browser, as an in-depth case study, may be found in Stephanidis et al. (2000).

## 6. Summary and conclusions

This paper has presented the unified user interface design method in terms of primary objective, underlying process, representation, and design outcomes. Unified user interface design is intended to enable the ‘fusion’ of potentially distinct design alternatives, suitable for different user groups and contexts of use, into a single unified form, as well as to provide a design structure which can be easily translated into a target implementation. By this account, the method is considered especially relevant for the design of systems that are required to exhibit adaptable and adaptive behavior, in order to support individualization to different target user groups and usage-contexts. In such interactive applications, the design of alternative dialogue patterns is necessitated due to the varying requirements and characteristics of end-users.

In terms of process, the method postulates polymorphic task decomposition as an iterative engagement through which abstract design patterns become specialized to depict concrete alternatives suitable for the designated situations of use. Through polymorphic task decomposition, the unified user interface design method enables designers to investigate and encapsulate adaptation-oriented interactive behaviors into a single design construction. To this effect, polymorphic task decomposition is a prescriptive guide of what is to be attained, rather than how it is to be attained, and thus it is orthogonal to many existing design instruments.

The outcomes of unified user interface design include the polymorphic task hierarchy and a rich design-space which provides the rationale underpinning the context-sensitive selection amongst design alternatives. A distinctive property of the polymorphic task hierarchy is that it can be mapped into a corresponding set of specifications from which interactive behaviors can be generated. This is an important contribution of the method to HCI design, since it bridges the gap between design and implementation, which has traditionally challenged user interface engineering.

It is argued that interaction design becomes increasingly a knowledge-intensive endeavour. 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. To this end, analytical design methods, such as unified user interface design, are anticipated to become necessary tools for capturing and representing the global design context of interactive products and services. Moreover, adaptation is likely to predominate as a technique for addressing

the compelling requirements for customization, accessibility and high quality of interaction. Thus, it 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.

The unified user interface design method has been applied, tested, refined, formulated and documented in its present form through a number of research projects partially funded by the European Commission (TP1001, ACCESS;<sup>1</sup> ACTS AC042, AVANTI;<sup>2</sup> IST-1999-20656, PALIO;<sup>3</sup> IST-1999-14101, IS4ALL<sup>4</sup>), or national government funding (EPET-II: NAUTILUS<sup>5</sup>). Also, it has been taught in various tutorials (Stephanidis et al., 1997, 1999, 2001b,c), attracting numerous participants from industry and academia, while stimulating fruitful discussions and exchange of ideas, that in many cases helped in enhancing different practical aspects of the original method.

## Acknowledgements

The ACCESS TP1001 (Development platform for unified ACCESS to enabling environments) project was partially funded by the TIDE Programme of the European Commission, and lasted 36 months (January 1, 1994 to December 31, 1996). The partners of the ACCESS consortium are: CNR-IROE (Italy)—Prime contractor; ICS-FORTH (Greece); University of Hertfordshire (United Kingdom); University of Athens (Greece); NAWH (Finland); VTT (Finland); Hereward College (United Kingdom); RNIB (United Kingdom); Seleco (Italy); MA Systems and Control (United Kingdom); PIKOMED (Finland).

The AVANTI AC042 (Adaptable and Adaptive Interaction in Multimedia Telecommunications Applications) project was partially funded by the ACTS Program of the European Commission, and lasted 36 months (September 1, 1995 to August 31, 1998). The partners of the AVANTI consortium are ALCATEL Italia, Siette division (Italy)—Prime Contractor; IROE-CNR (Italy); ICS-FORTH (Greece); GMD (Germany), VTT (Finland); University of Siena (Italy), MA Systems and Control (UK); ECG (Italy); MATHEMA (Italy); University of Linz (Austria); EUROGICIEL (France); TELECOM (Italy); TECO (Italy); ADR Study (Italy).

## References

- Bellotti, V., 1993. Integrating theoreticians' and practitioners' perspectives with design rationale, Proceedings of the INTERCHI'93 Conference on Human Factors in Computing Systems, Amsterdam, Netherlands, pp. 101–106.

<sup>1</sup> Development Platform for Unified Access To Enabling Environments.

<sup>2</sup> Adaptive and Adaptable Interactions for Multimedia Telecommunications Applications.

<sup>3</sup> Personalized Access to Local Information and Services for Tourists.

<sup>4</sup> Information Society for All.

<sup>5</sup> Unified Web Browser for People with Disabilities.

- Butler, K., Esposito, C., Klawitter, D., 1997. Designing more deeper: integrating task analysis, process simulation and object definition, *Proceedings of the ACM DIS'97 Symposium on Designing Interactive Systems*, Amsterdam, Netherlands, pp. 37–54.
- Desoi, J., Lively, W., Sheppard, S., 1989. Graphical specification of user interfaces with behavior abstraction, *Proceedings of the CHI'89 Conference on Human Factors in Computing Systems*, Austin, TX, pp. 139–144.
- Hartson, R., Hix, D., 1989. Human–computer interface development: concepts and systems for its management. *ACM Computing Surveys* 21 (1), 241–247.
- Hartson, H.R., Siocchi, A.C., Hix, D., 1990. The UAN: a user-oriented representation for direct manipulation interface design. *ACM Transactions on Information Systems* 8 (3), 181–203.
- Hill, R., 1986. Supporting concurrency, communication and synchronisation in human–computer interaction—the Sassafra UIMS. *ACM Transactions on Graphics* 5 (3), 289–320.
- Hoare, C.A.R., 1978. Communicating sequential processes. *Communication of the ACM* 21 (8), 666–677.
- Johnson, P., Johnson, H., Waddington, P., Shouls, A., 1988. Task-related knowledge structures: analysis, modeling, and applications. In: Jones, D.M., Winder, R. (Eds.), *Cambridge University Press*, pp. 35–62.
- Kandle, H., 1995. Integration of scenarios with their purposes in task modelling, *Proceedings of the ACM DIS'95 Symposium on Designing Interactive Systems*, MI, USA, pp. 227–235.
- Kneer, B., Szwillus, G., 1995. OBSM: a notation to integrate different levels of user interface design, *Proceedings of the ACM DIS'95 Symposium on Designing Interactive Systems*, MI, USA, pp. 25–31.
- Manheimer, J., Burnett, R., Wallerns, J., 1990. A case study of user interface management system development and application, *Proceedings of the CHI'90 Conference on Human Factors in Computing Systems* (Seattle, Washington), *ACM Press*.
- Marcus, A., 1996. Icon design and symbol design issues for graphical interfaces. In: Del Galdo, E., Nielsen, J. (Eds.), *International User Interfaces*, Wiley, New York, pp. 257–270.
- McLean, A., McKerlie, D., 1995. Design Space Analysis and Use-Representations, Technical Report EPC-1995-102, Rank Xerox.
- Olsen, D., 1990. Propositional production systems for dialog description, *Proceedings of the ACM CHI'90 Conference on Human Factors in Computing Systems*, pp. 57–63.
- Payne, S., 1984. Task-action grammars, *Proceedings of IFIP Conference on Human–Computer Interaction: INTERACT'84*, vol. 1. North-Holland/Elsevier Science, London/Amsterdam, pp. 139–144.
- Potts, C., 1995. Using schematic scenarios to understand user needs, *Proceedings of the ACM DIS'95 Symposium on Designing Interactive Systems*, MI, USA, pp. 247–256.
- Royer, T., 1995. Using scenario-based designs to review user interface changes and enhancements, *Proceedings of the ACM DIS'95 Symposium on Designing Interactive Systems*, MI, USA, pp. 277–246.
- Saldarini, R., 1989. Analysis and design of business information systems, *Structured Systems Analysis*, MacMillan Publishing, New York, pp. 22–23.
- Sary, C., 1996. Integrating workflow representations into User Interface design representations, *Software Concepts and Tools*, vol. 17.
- Stephanidis, C., Savidis, A., 2001. Universal access in the information society: methods, tools and interaction technologies. *Universal Access in the Information Society* 1 (1), 40–55.
- Stephanidis, C., Savidis, A., Akoumianakis, D., 1997. Tutorial on 'Unified Interface Development: Tools for Constructing Accessible and Usable User interfaces', Tutorial no 13 in the Seventh International Conference on Human–Computer Interaction (HCI International '97), San Francisco, USA.
- Stephanidis, C., Akoumianakis, D., Paramythis, A., 1999. Tutorial on 'Coping with Diversity in HCI: Techniques for Adaptable and Adaptive Interaction', Tutorial no 11 in the Eighth International Conference on Human–Computer Interaction (HCI International '99), Munich, Germany.
- Stephanidis, C., Paramythis, A., Sfyarakis, M., Savidis, A., 2001a. A case study in Unified User Interface development: the AVANTI Web Browser. In: Stephanidis, C., (Ed.), *User Interfaces for All—Concepts, Methods, and Tools*, Lawrence Erlbaum Associates, Mahwah, NJ, pp. 525–568.
- Stephanidis, C., Savidis, A., Akoumianakis, D., 2001b. Tutorial on 'Engineering Universal Access: Unified User Interfaces', Tutorial in the First Universal Access in Human-Computer Interaction Conference

- (UAHCI 2001), jointly with the Ninth International Conference on Human–Computer Interaction (HCI International 2001), New Orleans, LA, USA.
- Stephanidis, C., Savidis, A., Akoumianakis, D., 2001c. Tutorial on ‘Universally accessible UIs: the unified user interface development’, Tutorial in the ACM Conference on Human Factors in Computing Systems (CHI 2001), Seattle, Washington.
- Wilson, S., Johnson, P., 1995. Empowering users in task-based approach to design, Proceedings of ACM DIS ’95 Symposium on Designing Interactive Systems, MI, USA, pp. 25–31.