# Designing user-adapted interfaces: the unified design method for transformable interactions

*A. Savidis, A. Paramythis, D. Akoumianakis and C. Stephanidis*
Institute of Computer Science,
Foundation for Research and Technology-Hellas (FORTH),
Science and Technology Park of Crete,
P.O. Box 1385, GR 711 10, Heraklion, Crete, Greece,
Tel: +30-81-391741, Fax: +30-81-391740
E-mails: {as, alpar, demosthe, cs}@ics.forth.gr

**ABSTRACT**
In the interface design process, diverse user requirements and characteristics lead to alternative dialogue patterns. User-adapted interfaces, capable of self-adapting to individual end-user requirements, should encompass alternative dialogue components into a single implementation form. The process of designing user-adapted interactive applications necessarily engages the manipulation of alternative design artifacts, while for the implementation process a single design is needed, as opposed to alternative design versions. The unified design method is targeted towards the organization of alternative design artifacts into a single representation structure. Relationships among alternative artifacts in user-adapted design, such as exclusion, compatibility, augmentation and substitution, need to be explicitly represented.

**KEYWORDS:** Artifact-oriented design methodologies, user-adapted interaction, User Interfaces for All, polymorphic task hierarchies, task-oriented design.

## INTRODUCTION

A variety of design approaches have been defined in the past, such as hierarchical task decomposition [8] or GOMS analysis [3]. The main purpose of the design information consolidated during the design process is to support subsequent development phases, like target implementation and / or usability evaluation. In this context, driven from the objective of automating or accelerating the transition between the design and implementation or evaluation phases, various alternative design approaches emerged, some of which have been provided with tool support for automatically realizing the implementation and evaluation phases. Examples of such approaches are Task-Action Grammars (TAGs) [13], asynchronous models of user actions like CSP [7], the User-Action Notation (UAN) [5], propositional production systems (PPS) [12], variants or hybrids of the original task model such as TADEUS [16], etc. It is argued that new User Interface design methods emerge 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 normally realized in the context of a task analysis design process. Hence, if the explicit representation of graphical constraints is necessary (e.g. as an input to the implementation phase), a dedicated design process must be executed for the extraction of such necessary design information. In situations where a single design approach does not fulfil the requirements of the design process, the fusion of alternative techniques is applied, leading to hybrid design methodologies (e.g. combination of task analysis and graphic design methods).


DIS '97 Amsterdam, The Netherlands


The work reported in this paper has been carried out in the context of realizing the User Interfaces for All objective [17]. Following this objective, interactive applications should be designed and implemented in a way ensuring that potentially all user categories are given equal access and maintaining high quality of interaction. The ability for automatic interface individualization / adaptation to the requirements of individual end-users is of key importance in this context, demanding methods for designing and implementing interfaces that *adapt themselves* (henceforth we will use the term user-adapted to denote this type of interface behaviour). It will be shown that existing design approaches do not suffice for designing user-aware interactive applications which can be automatically adapted to individual users; the unified design method is proposed in order to address this issue, building upon the foundations of hierarchical task decomposition.

### The transformable interface concept

In general terms, a user-adapted interface is aware of key user attributes and utilises such knowledge in order to realize appropriate interaction with each individual end-user (i.e. automatic adaptation process). End-users exhibiting diverse attribute values are likely to require radically different dialogue styles to be chosen. Hence, a user-adapted interface seems to perform a transformation process for each individual end-user, relying primarily upon pre-recorded user

information; for this reason, the term transformable interface will be used, reflecting the behavioural dynamics of a user-adapted interface. In the context of user-adapted interaction realizing the goal of User Interfaces for All, there is no "average" user. This gives rise to two issues: (i) a transformable interface implementation should realize more than a single interface design, since it is a user-centred process; and (ii) enumerating all alternative designs is practically not realistic, since their number is directly related to all possible combinations of user attribute values (cause of differentiation among the various design decisions).

Existing design approaches support design processes which lead to a single artefact. The ability to differentiate and polymorphose is not embedded within available design techniques, thus requiring explicit enumeration of the alternative designs, in order to support representation of the numerous alternative interactive "faces" that a user-adapted interface is able to realize (due to the transformation capability); this inability to unify alternative final design decisions introduces technical problems when the resulting designs needs to be implemented as a single interactive system. The unified interface design method has been defined in order to: (i) enable expression of the alternative enumerated designs of a user-adapted interface into a single unified form, without requiring explicit enumeration; and (ii) provide a design model for organizing dialogue patterns which can be easily translated to a software organization model for the required interface implementation.

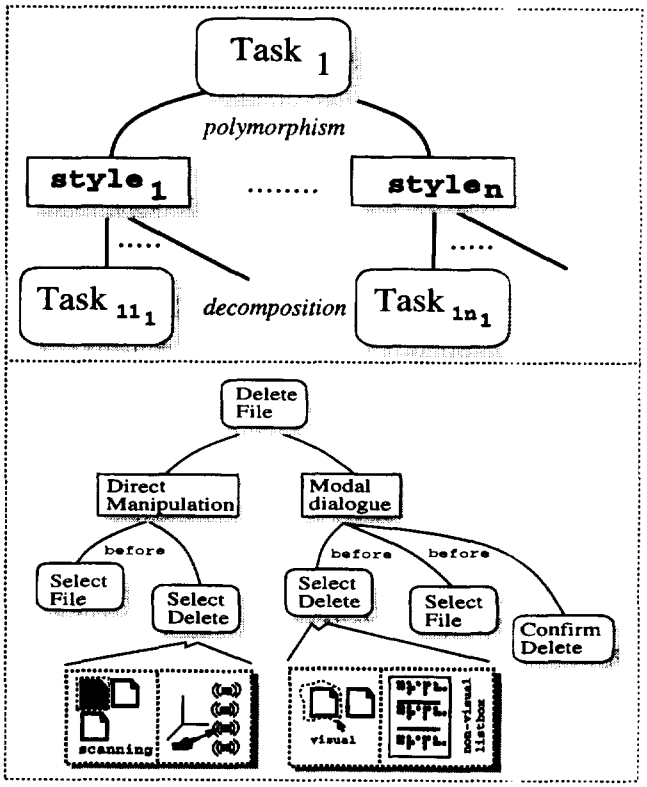


Figure 1: The polymorphic task hierarchy concept.

**Brief overview of the unified design approach**

The unified interface design method builds upon the hierarchical task analysis method. It introduces the notion 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 user tasks, starting from the abstract level, by incrementally specializing in a polymorphic fashion (since different design alternatives are likely to be associated with differing user attribute values), towards the physical level of interaction. In a more compact definition, the unified design process drives an "abstract task definition with incremental polymorphic physical specialization".

## POLYMORPHIC TASK HIERARCHIES

The polymorphic task hierarchy decomposition method combines three fundamental properties: (a) hierarchical organization, (b) polymorphism, and (c) task operators. The hierarchical decomposition adopts the original properties of hierarchical task analysis 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-centred design requirements. Finally, task operators, which are based on the powerful CSP language for describing the behaviour of reactive systems [7], enable the expression of dialogue control flow formulas for task accomplishment; these operators are *before* (sequencing), *or* (parallelism), *xor* (exclusive completion), * (simple repetition) and + (absolute repetition). The concept of polymorphic task hierarchies is illustrated in Figure 1. Each alternative task decomposition is called a decomposition style, or simply a *style*, and is given an arbitrary name; the alternative task sub-hierarchies are attached to their respective styles. The example in Figure 1 shows how two alternative dialogue styles for a "Delete File" task can be designed: one exhibiting direct manipulation properties (i.e. "Direct Manipulation" style, file object is selected prior to operation to be applied) and another realizing modal dialogue (i.e. "Modal dialogue" style, the delete operation is selected, followed by target file selection, followed by confirmation of operation). Additionally, the example demonstrates the case of physical specialization. Since "selection" is an abstract task, it is possible to design alternative ways for physically instantiating the selection dialogue (see Figure 1, lower-part): via scanning techniques for motor-impaired users, via 3D hand-pointing on 3D-auditory cues for blind people, via enclosing areas for sighted able users, and via Braille output and keyboard input for deaf-blind users.

The unified design method does not impose the designer to follow the polymorphic task decomposition all the way down the user-task hierarchy, until primitive actions are met. At any level, a non-polymorphic task can be specialized following any design method chosen by the interface designer; for instance, in Figure 1 (lower-part) graphical

illustrations are used to describe each of the alternative physical instantiations of the abstract "selection" task. Similarly, the interface designer may choose a more suitable model for describing user actions for device-level interaction (e.g drawing, drag-and-drop, concurrent input) than the CSP-based operators which have been primarily chosen for expressing task relationships, such as an event-based representation (e.g. ERL [6], UAN [5]).

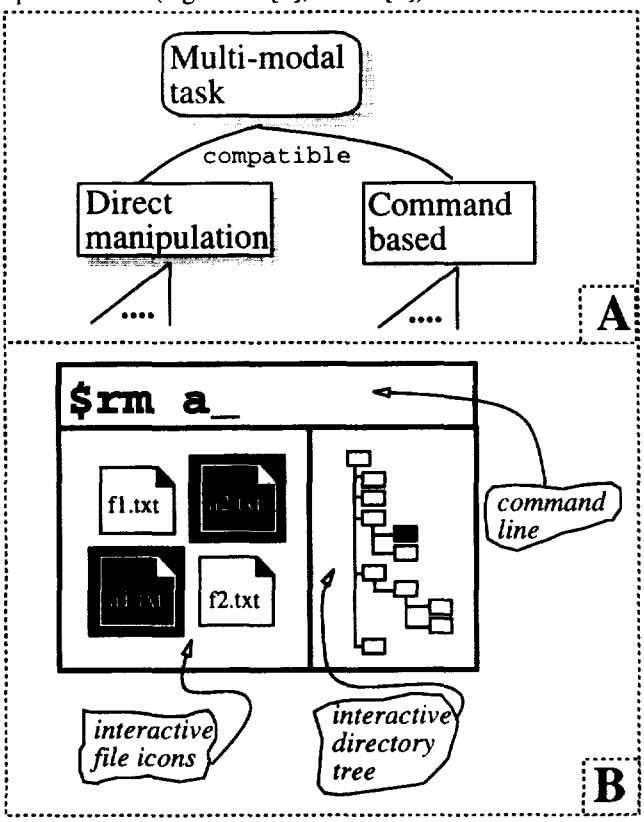


Figure 2: Multimodality in conjunction to task-level polymorphism.

The polymorphic task decomposition method is also appropriate in cases where the design process reveals the necessity of having multiple alternative sub-dialogues available concurrently to the user, for performing a particular single 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. In our example of Figure 2 (part A), a task is polymorphosed with two alternative sub-hierarchies (i.e styles), a direct manipulation and a command-based respectively; a physical specialization design scenario is also provided in Figure 2 (part B). These two styles are defined to be compatible, which implies that they may coexist at run-time (i.e. the end-user may freely use the command-line or the interactive file manager interchangeably for file manipulation). In general, combining alternative decomposition styles may result in high quality interaction, if the user can benefit from the specific advantages of each distinct style. For instance, in the example of Figure 2 (part B), the user may choose the command-line for moving a file to another folder, while he / she may prefer the interactive file manager for direct manipulation deletion of a folder (e.g. click on folder icon and then press "del" key, perform "X" gesture with a pointing device over the target folder).

**The polymorphic task decomposition process**

The abstract task definition and polymorphic physical specialization is a recursive process which involves abstract / physical tasks with conventional / polymorphic decompositions. The overall process is illustrated in Figure 3. The decomposition process starts from abstract or physical tasks (depending on whether top-level user tasks can be defined as abstract or not). The transitions from each of the four states are described below (see also Figure 3).

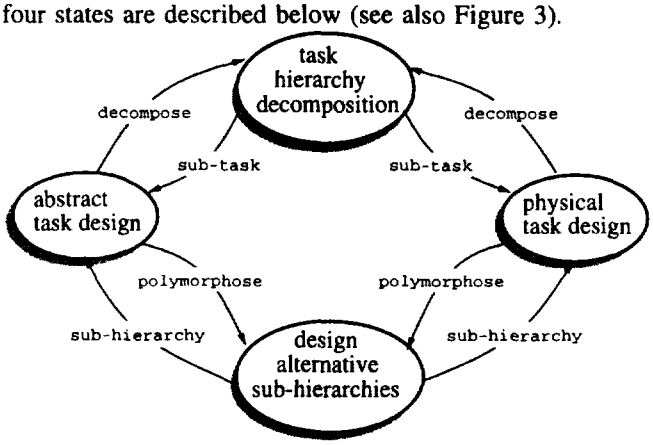


Figure 3: The polymorphic task decomposition process.

*From "abstract task design" state:* An abstract task can be decomposed either in a polymorphic fashion (if design requirements pose the necessity for alternative dialogue patterns) or in a conventional manner, following a single decomposition scheme. In the case of a single decomposition scheme, the transition is realized via a "decomposition" action, leading to the "task hierarchy decomposition" state. In the case of polymorphic decomposition, the transition is realized via a "polymorphose" action, leading to the "design alternative sub-hierarchies" state.

*From "task hierarchy decomposition" state:* The sub-tasks identified need to be further decomposed. For each such sub-task, if it is an abstract task, there is a "sub-task" transition to the "abstract task design" state, else (it is a physical task) a similar transition to the "physical task design" state.

*From "design alternative sub-hierarchies" state:* The sub-tasks from the top level of each sub-hierarchy defined (the sub-hierarchy is incrementally built level-by-level) need to be further decomposed. For each alternative sub-hierarchy, and for each top-level task, if it is an abstract task, there is a "sub-hierarchy" transition to the "abstract task design" state, else (it is a physical task) a similar transition to the "physical task design" state.

*From "physical task design" state:* The transitions executed from this state are exactly identical to those from the "abstract task design" state.

Example. The polymorphic task decomposition for the example of Figure 1 will be briefly discussed, in view of the process model shown within Figure 3; the sequence of steps is illustrated under Figure 4 (states are mentioned with brief names). The initial state is "abstract state" (step 1), since "Delete File" is an abstract task. Through the "polymorphose" transition, two alternative sub-hierarchies result (step 2). For each sub-hierarchy, and for each top-level task, we identify whether it is abstract or physical in order to continue the process (step 3). For instance, both "Select File" and "Select Delete" are abstract tasks (the rest are not shown for clarity). We show the next steps for the "Select Delete" task, which is polymorphosed (step 4), resulting in two sub-hierarchies (one visual dialogue and another for non-visual dialogue). The top-level tasks for each of the two sub-hierarchies are in this case physical (step 5, as opposed to step 3, where all tasks are abstract). The "visual rubber-banding" task is sub-sequently decomposed (step 6) to a conventional task hierarchy (step 6 could be realized via any other approach, if required, such as an event modelling technique).

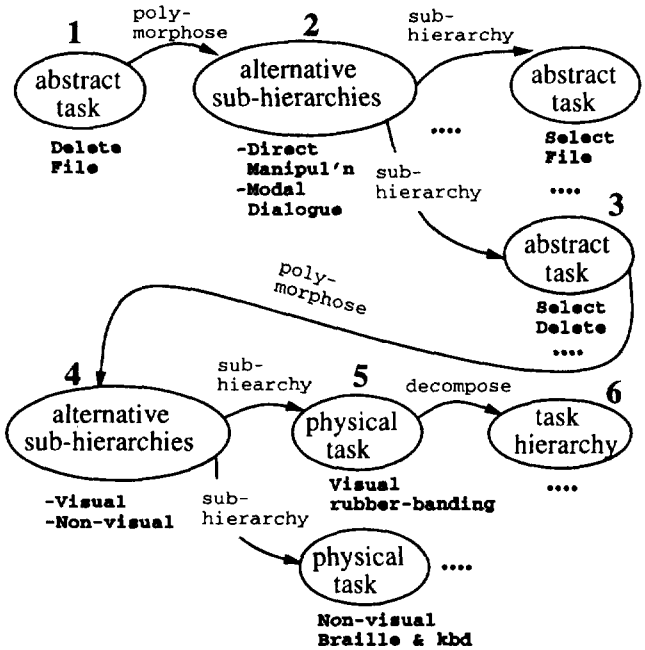


Figure 4: A decomposition process example.

## DESIGNING ALTERNATIVE STYLES

The polymorphic task model provides the technique for organizing all the alternative dialogue patterns of a user-adapted interface into a unified form. Such a hierarchical structure realizes the fusion of the various alternative designs which can be enumerated for a given user-adapted interface. Apart from the polymorphic organization model, the following key issues need to be addressed next: (i) when polymorphism should be applied; (ii) which are the user attributes that need to be considered; and (iii) how the design rationale connecting the designed styles with the user attributes is documented.

*Identifying levels of potential polymorphism.* In the context of the unified 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 Figure 3) fulfils all the combinations of the target user attribute values. More practically, in case that at any step of the task analysis process there is a particular decomposition which does not address some combinations of the target user attribute values, then the specific end-users with those attribute values are not considered at this step of the design process. Hence, a level of potential polymorphism has been identified and the design gap may be closed by constructing the necessary alternative sub-hierarchy(-ies) addressing the excluded user attribute values.

*Constructing the space of user attributes.* In the unified design method, the end-user representation technique as a finite set of attribute values has been chosen because of its genericity, since it does not pose restrictions on the employment of any particular user modelling approach for the implementation process. There is no predefined / fixed 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. Also, the value domains for each attribute class are chosen by interface designers and human-factors experts as part of the design process (the value sets need not be finite). The broader the set of values, the higher the differentiation capability among various individual end-users; for instance, commercial systems realizing a single design for an "average" user have zero differentiation capability.

| **Task**: Delete File | |
|---|---|
| **Style**: Direct Manipulation | **Style**: Modal dialogue |
| **Users**: Expert, Frequent, Average. | **Users**: Casual, Naive. |
| **Targets**: Speed, naturalness, flexibility. | **Targets**: Safety, guided steps. |
| **Properties**: Object first, function next. | **Properties**: Function first, object next. |
| **Combinations**: Exclusive. | **Combinations**: Exclusive. |

Table 1: Example of recording design rationale.

*Recording design rationale 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 the design logic for deciding alternative styles, by directly associating user parameters and

design goals with the constructed artefact (style). The parameters are: (i) *users* (specific user attribute values addressed by a style); (ii) *targets* (concrete design targets for a style); (iii) *properties* (the specific differentiating interaction properties of a style, in comparison to other styles); and (iv) *combinations* (compatibility and relationships with other styles). The values of these parameters are recorded during the decomposition process for each style in the form of a table. In Table 1, an example is shown for the definition of these parameters regarding the two alternative styles for the "Delete File" task (see Figure 1). The notion of design rationale, as used in the context of the unified design method has 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 [2] before reaching final design decisions, while in the our case, design rationale records the different user attributes and design objectives underpinning the already made (i.e. final) design decisions. The set of five parameters previously defined serves mostly as an "indexing" method for organizing final design decisions with primary keys the "Task" and "Users" parameters. The outcome of the unified design approach is a single hierarchical artifact, composed of user-oriented final design decisions associated to directly computable parameters (i.e. task and user attributes). The key advantage is that this resulting artifact realizes by itself a concrete implementation model for user-adapted interactions.

## ENGAGING ABSTRACT INTERACTION OBJECTS IN THE UNIFIED DESIGN METHOD

During the task decomposition process, some sub-tasks can be directly related to user-input actions which can be managed via interaction objects. For instance, selecting from a list of options, interactively changing the state of a boolean parameter, providing an arithmetic value, etc, are typical examples of input tasks which can be realized via the predefined dialogues implemented by various interaction objects. In such cases, it is desirable to employ general / abstract object classes, in order to enable alternative physical object classes to be selected for differing user-, design- and domain- requirements. Past approaches for abstract object categories exist, such as input interaction tasks [4], meta-widgets [19], virtual objects [14], and more implementation-oriented models such as interactors [11]. It is argued that designers primarily think in terms of specific instances and physical interface scenarios, especially if the task analysis and graphic design processes are carried out by different teams, rather than composing interface components via abstract behaviours and objects. In this context, we have defined a role-based model (see Figure 5) for "filtering" already made design decisions in order to identify "points" in which abstract interaction objects can be employed in the design representation. Three role categories for interaction objects are identified:

*Lexical role*, in which case the interaction object is employed for appearance / presentation needs. If such a role can be applied independently of physical realization, then an abstraction can be identified. For example, assume a "Message" interaction object, which has only one attribute defining the message content (e.g. a string). It should be noted that the content could be verbal (i.e. the string is a phrase), if the user understands natural language, or even symbolic (i.e. the string is a file name where a symbolic sequence is stored), if the user understands symbolic languages. The presentation properties (e.g. emphasizing with an icon, other visual / auditory effects) concern the physical implementation that may have alternative realizations.

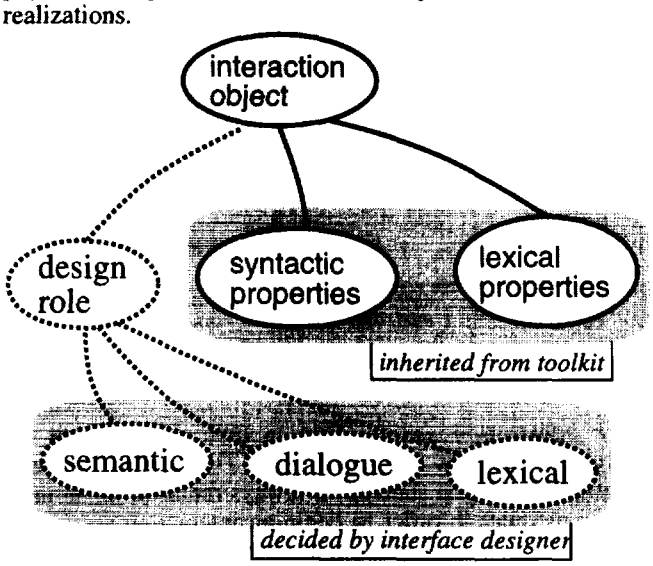


Figure 5: Role-based model of interaction objects.

*Syntactic / dialogue role*, in which case the object serves a specific purpose in the design of dialogue sequencing. If the role can be applied independently of physical realization, then an abstraction can be identified. For example, a "continue" button, a "confirm" button or a button to initiate an operation, play the role of a "Command" given by the user, in the particular dialogue context. Such a "Command" class may be applied to provide, for instance, execution, confirmation, cancellation or progress and may be applicable for various metaphors. It could be physically realized as a conventional push-button for the Desk-top metaphor, a voice-input command object for non-visual interaction, and a particular symbol for language-impaired users. The abstract interaction object "Command" may have only one boolean attribute to control whether it is accessible or not; the presentation feedback for indicating accessibility status, could be different, depending on its physical realization.

*Semantic role*, where an interaction object interactively realizes a domain object. For instance, an interaction object may present a domain object content or provide the means to enable "editing" of the content by the user. In such cases, it is always possible to transform the role to a proper abstract class. A typical example is the provision of a

numeric value by the user. A "Valuator" abstract object could be defined for this purpose, having various properties related to the type of numeric value required (e.g. range, discrete or real).

## RUN-TIME ARCHITECTURAL PROPERTIES OF TRANSFORMABLE INTERFACES

In Figure 6, an architectural model for transformable interfaces is outlined. This model consists of independent processes / components relying upon the design information which results from the unified design process. The role of each component follows:

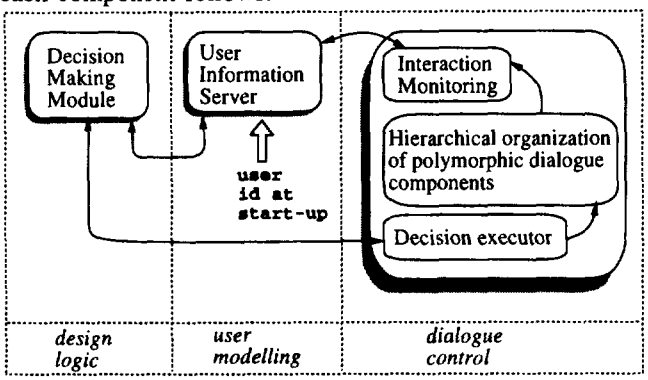


Figure 6: Run-time model for transformable interfaces.

*User information server.* This module encompasses the individual profiles of end-users. The initial end-user identification (upon start-up) is always required (as an input to this module) in order to begin an interaction session. In case of desk-top systems with dedicated users, all interactive applications may gain a fixed "user id". However, since today the trend is to enable users to access computer applications from virtually everywhere, explicit user identification will be required (i.e. support nomadic use). The user information server will map the user id to the corresponding user profile; this profile is a sequence of user attribute values. The user information server may employ other additional knowledge-based components for processing such user profiles, making additional assumptions about the user or even updating the user profile attribute values; for instance, systems like BGP-MS [9], PROTUM [18], or USE-IT [1] could be employed for such intelligent processing purposes. Apart from the initial manipulation of user profiles, the user information server may process at run-time interaction monitoring information for drawing additional conclusions about the user. Such conclusions may concern: dynamic assumptions of user interests, loss of orientation in performing certain tasks, fatigue, inability to complete a task, etc.

*Decision making module.* This module realizes the logic for activating the necessary styles, on the basis of the user attribute values received from the user information server (possibly after additional processing and inferences applied by the user information server). The decision making logic primarily realizes recorded design rationale (in the form of tables) during the polymorphic task decomposition process, in a rule-based knowledge representation form (e.g. "if user attribute X has value Y then for polymorphic task T style S is active"). The user information server will initially export the user attribute values to the decision making module, thus resulting on initial style decisions exactly before interaction starts. During interaction, further updates on particular parameters or dynamic assumptions made about certain user attributes (at the user information server) will be also sent to the decision making module. This process may result in decisions changing some style decisions made at start-up, thus dynamically changing aspects of the User Interface. All drawn decisions are directly communicated to the User Interface implementation module.

*Dialogue control.* This module implements all the various dialogue patterns identified during the polymorphic task decomposition process. The implementational organization of the dialogue components should enable externally originated style activation decisions to be directly applied. The resulting implementation may reflect the hierarchical organization of the dialogue components in the context of the polymorphic task model. Apart from dialogue components, there are two other layers of functionality: (i) the decision executor, which communicates with the decision making module (receives decisions or dynamically requires decision making); and (ii) the monitoring components which send interaction monitoring data to the user information server for further processing (e.g. key-strokes, notifications for use of interaction objects, task-level monitoring - level and frequency of monitoring depends on the capabilities of the employed user modelling tool within the user information server).

### Adaptability and adaptivity: the two faces of the transformation process

The run-time interface transformation process in user-adapted interaction can be seen as a combination of two complementary classes of system initiated actions:
(i) adaptation decisions driven from initial knowledge on user attribute values available at start-up (i.e. what the information server knows about the user prior to interaction), and (ii) adaptation decisions drawn due to knowledge on user attribute values inferred during interaction (i.e. assumptions regarding the user, made by the user information server on the basis of interaction monitoring information).

The former behaviour has been attributed as *adaptability*, reflecting the interface capability to automatically tailor itself initially to each individual end-user. The latter behaviour has been defined as *adaptivity*, and characterizes the interface capability to cope with the dynamically (during interaction) changing / emerging user requirements. Past work has mainly addressed the issue of adaptivity [15], while more recent work, initially motivated by the need of interface accessibility by disabled people, has focused on adaptability [1]. In the context of accessibility, the adaptability behaviour is of higher importance, since the essence is to initially

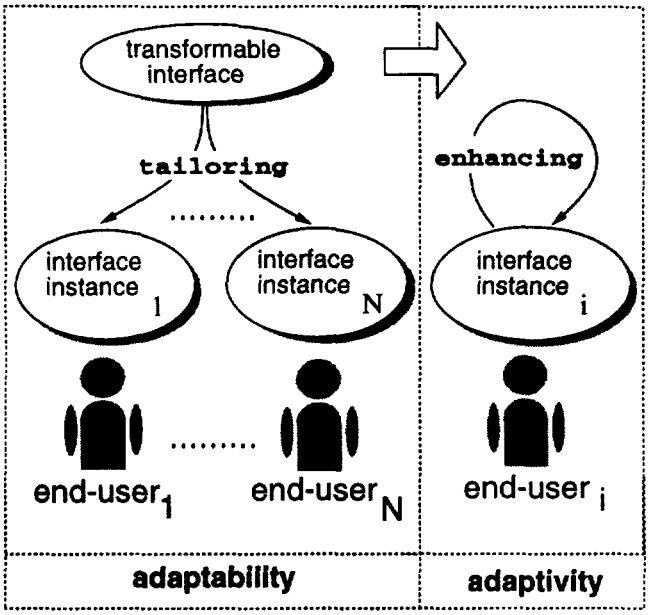


Figure 7: The complementary roles of adaptability and adaptivity in transformable interfaces.

realize an interface instance that is appropriate (i.e. also accessible) for the end-user. Instead, adaptivity is applied on an accessible running interface (i.e. user performs interaction), since interaction monitoring is required for the identification of changing / emerging requirements that will drive dynamic interface enhancements. The complementary role of adaptability and adaptivity approaches is illustrated within Figure 7.

## THE INTERFACE DIFFERENTIATION EFFECTS FROM TASK-LEVEL POLYMORPHISM

During the unified design process, polymorphism may be applied at any level of the task hierarchy and for any task. The various behavioural / morphological differences which can be observed among the specific interface instances resulting from the alternative sub-hierarchies for a given polymorphic task, depend on the level of this task within the overall hierarchy. Such evident variations among alternative interfaces instances (due to the transformation behaviour) are characterized as differentiation effects. Three main categories of such interface differentiation effects are distinguished, on the basis of the level at which polymorphism is applied (normally, combinations of these effects will be present):

*Polymorphism on top-level tasks.* These tasks, which belong at the highest levels of the hierarchy and are called overall / main / top-level user tasks, concern what the user has to accomplish with an interactive application (see Figure 8, - part A); for instance, edit a document, send an e-mail, perform spell-checking, construct graphic illustrations, etc. When polymorphism is applied at this level, the interface instances are likely to be effected by structural differences, providing the feel of alternative versions of the same interactive environment (i.e. effect seems global).

*Polymorphism on intermediate tasks.* These tasks belong to the middle hierarchy levels, and concern particular dialogue states reachable after performing some required sets of actions (see Figure 8 - part B); for instance, dialogue boxes for setting parameters, executing selected operations, editing retrieved items, etc. The effect on alternative interface instances is to have overall similarities, with localized differences in interactive components and intermediate sub-dialogues.

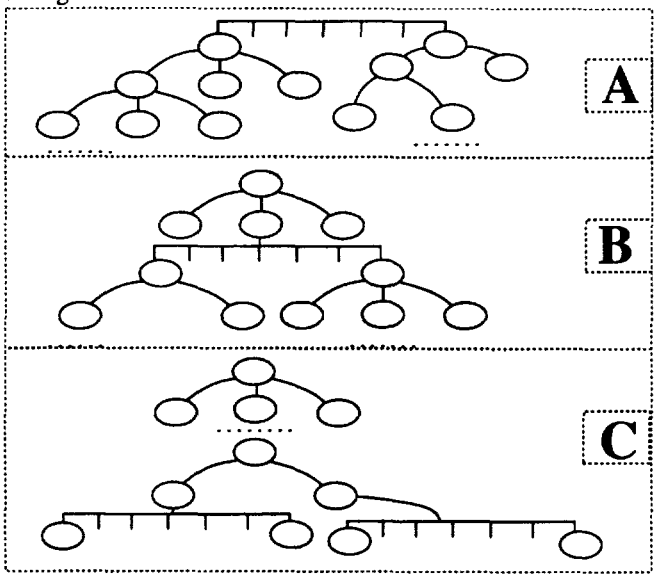


Figure 8: Levels of task-based polymorphism.

*Polymorphism on primitive tasks.* These tasks appear at the lowest levels of the task hierarchy (leaves, see Figure 7 - part C) and concern primitive user actions (i.e. actions which can be directly supported by the primitive physical interaction elements); for instance, pressing a button, moving a slider, defining a stroke with the mouse, etc. The polymorphism at this level causes differences on device-level input syntax and / or on the type of interaction objects in some interface components.

## APPLYING THE UNIFIED DESIGN METHOD FOR AN ADAPTABLE AND ADAPTIVE WEB BROWSER

The application of the unified interface design process will be discussed in the context AVANTI Project (Adaptable and Adaptive Interaction in Multimedia Telecommunications Applications), funded by the ACTS Programme of the Commission of the European Union (DG XIII). Some key design artifacts will be discussed, in situations where polymorphism has been applied, for the design of an adaptable and adaptive Web browser (for HTML 3.2). The target user audience for the AVANTI project is: 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). Some selected design examples will be discussed, concerning specific tasks, through which the possible relationships among the alternative styles for a given polymorphic task will be revealed.

*Link selection task.* In all existing Web browser (e.g. Netscape Navigator™, Microsoft Explorer™ and SUN HotJava™), links in Web documents are activated by pressing the mouse left mouse button while the cursor resides within the area of a link. In Figure 8, the design of the link selection dialogue (for textual links) is shown, realizing polymorphic design. There are two steps in link selection, according to the new design: (a) actually selecting a link (which can be done in two alternative ways, S1 and S2); and (b) requiring confirmation for loading target document (also done in two ways, S3 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 rationale for polymorphism is presented in Figure 9 (only a brief summary is presented for clarity). The combinations between the designed styles are important (due to adaptivity, some dynamic style updates need to be explicitly mentioned):

- S1 mutually exclusive with S2.
- S3 mutually exclusive with Se
- S3 substitutes Se dynamically (if high error rates are detected)
- Se substitutes S3 dynamically (if in a satisfactory interaction history, link activation has been always followed by positive confirmation)

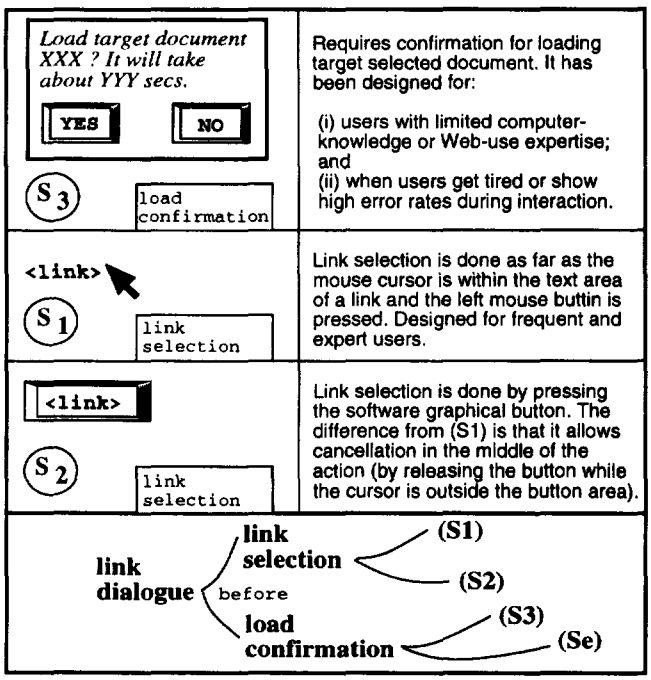

Figure 9: Link selection task decomposition.

*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, book-marking, load options). In Figure 10, two alternative designs are shown, primarily 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). These styles have the following relationships:

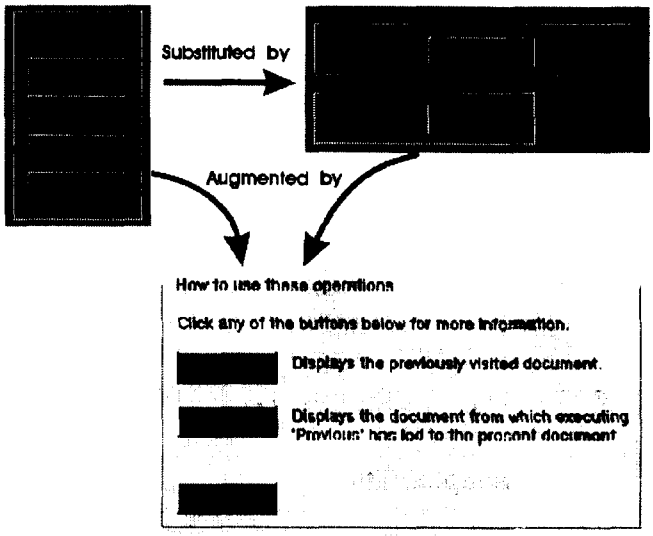

Figure 10: Alternative styles for page loading control.

- The top-left style is the style to be initially active (casual / naive values on application expertise).
- 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 (when the particular end-user completes a number of uses that will be considered as an advance to the "average" application expertise).
- The bottom style *augments* the particular active style, and it provides adaptive prompting / helping [15] for carrying out the operations in case inability to perform the task or high error rates are dynamically detected.

*Document browsing task.* The requirement to provide accessibility of the resulting browsers by motor-impaired users, necessitated the design of dialogue for enabling motor-impaired users explore 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 could have access on the original visual interface designed for able users, though via alternative input techniques. This approach proved to be very good for all tasks, except from the case of page browsing; 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

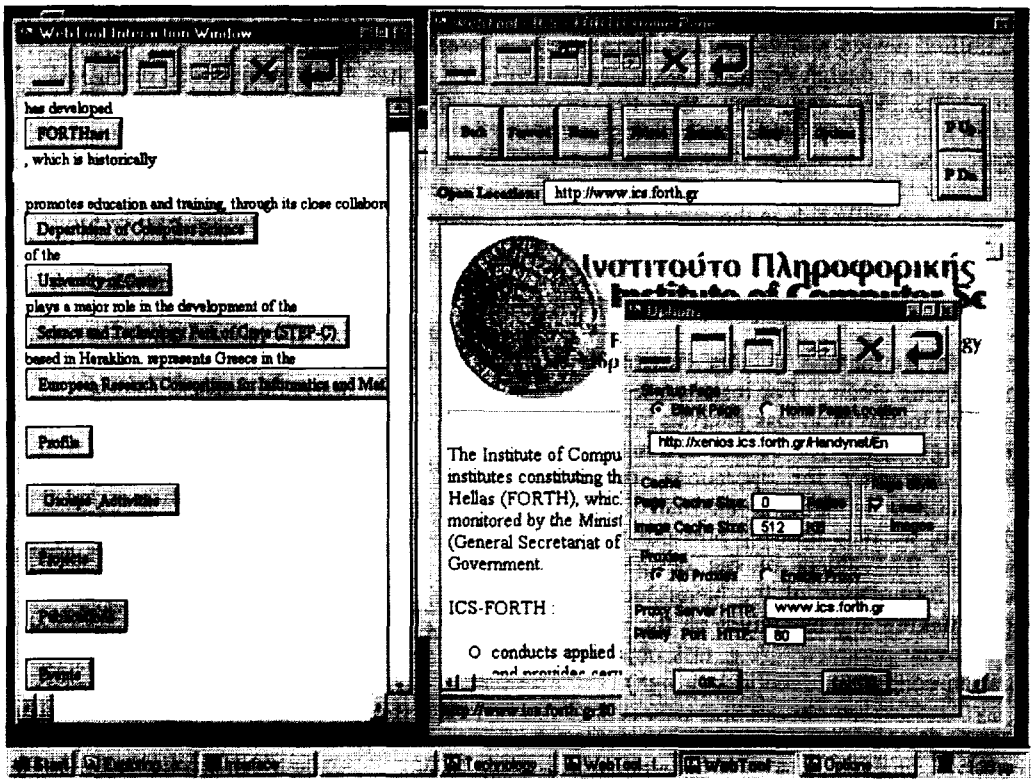

Figure 11: The window on the left provides the summary of links (i.e. all links collected and presented together). The document context for each link is also indicated, by presenting the previous and next lines of text that enclose each link.

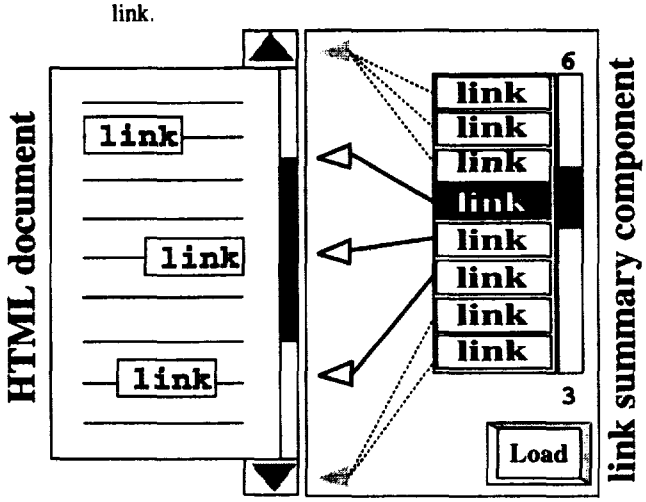


Figure 12: Link summary style. The document display on the left is automatically adjusted, so that the highlighted link is always visible. Dashed arrows indicate that document context including the associated link is above / below visible portion. Solid arrows are attached at visible links and point to the exact link position in the document visible area. The number displayed above / below the scrollbar of the links' summary listbox indicates how many more links are included above / below the first / last displayed link within the listbox.

information / links. This has resulted in the two alternative styles, illustrated in Figure 11 and Figure 12, which augment the page presentation style, and are mutually exclusive.

## THE UNIFIED DESIGN METHOD VERSUS DESIGN RATIONALE AND DESIGN SPACE ANALYSIS

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 [2] from multiple sources, such as design discussions and diverse kinds of theoretical analyses. QOC (Questions, Options & Criteria) [10] 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 the strengths and weaknesses [2].

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, however, they 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. This is fundamentally different in comparison to the unified design method, where alternative styles directly represent final design decisions associated to particular user attributes. We will show that the two approaches have different goals, provide methodologies at different level of specialization, however, they may perfectly work in a complementary fashion, combining the benefits of both approaches. Firstly, we will analyze the scope of design space analysis for diverse design problems and tasks.

**Design space analysis is a meta-model for cooperative design processes**

The design space analysis method is very flexible with respect to the subject being studied. For instance, it may be initiated during a typical hierarchical task analysis method in order to create a design space for certain artifacts (e.g. choosing the proper metaphor for operations, deciding appropriate data visualizations, judging alternative visual interface designs). Also, we may turn the previous process up-side down, so that the subject of the design space analysis could be the formulation of the hierarchical task model; for example, making argumentation on the abstract task hierarchy and relevant parameters (e.g. task objectives, relationships, initiation conditions, feedback design).

In this sense, there is theoretically no restriction on how systematic, exhaustive, analytic and multidisciplinary a design space analysis process may become. It is argued that the justification of such an unlimited scope of design space analysis techniques is due to its nature of being a powerful design-process meta-model, rather than a specific design methodology. Any systematic methodology for cooperative interface design may be instantiated through the specific regulations and dynamics of design space analysis. Moreover, it is believed that this meta-model is expressive enough to even manage the organization characteristics of the more general case of group-based decision making. This property enables design space analysis to be applied for virtually any problem domain; however, it should be used in conjunction with concrete and specialized design methods (many of which may be employed simultaneously within a single design project) when resolving specific design problems.

This is one good reason why, in practice, design space analysis techniques have been mainly employed in the context of scenario-based design, where the key properties are argumentation, assessment and exchange of ideas over a collection of multiple concrete design artifacts serving a common purpose. If special purpose design practices are needed, such as hierarchical task analysis, design verification (e.g. performed via model-checking techniques), device-level interaction design (e.g. employing event-based notations), etc, the design space analysis provides only the framework for communicating design ideas, while it clearly does not provide the concrete design methodologies required.

**Process-oriented versus artifact-oriented design methodologies**

As it has been previously discussed, the design space analysis technique constitutes a design process meta-model. Hence, it is primarily process-oriented and it gives particular emphasis on the organization of the design process, by means of a systematic communication and argumentation among designers, requiring well formulated and documented design suggestions.

| | |
|---|---|
| ***P1.*** | **Is an explicit strategy provided for the construction of design alternatives ?** |
| *UDM:* | Yes. The hierarchical polymorphic task analysis. |
| *DSA:* | No. The design space analysis is a meta-model for design processes. |
| ***P2.*** | **Are the design alternatives produced considered finalized / under discussion ?** |
| *UDM:* | All design alternatives are final design decisions. |
| *DSA:* | The design alternatives included are primarily under discussion / argumentation. |
| ***P3.*** | **What is the primary parameter giving birth to design alternatives ?** |
| *UDM:* | Different user attribute values. |
| *DSA:* | Design criteria. It is also natural that new design ideas precede the identification of their associated criteria / properties. |
| ***P4.*** | **How is the space of design alternatives related to the final design ?** |
| *UDM:* | All design alternatives (i.e. styles) are part of the final design outcome. |
| *DSA:* | The design space includes the final design, and it extensively encompasses additional information. |
| ***P5.*** | **How are design alternatives related to each other ?** |
| *UDM:* | Run-time relationships (exclusion, compatibility, substitution, augmentation). |
| *DSA:* | They have an arbitrary number of design relationships (criteria-based). |
| ***P6.*** | **What is the role of design rationale in the space of design alternatives ?** |
| *UDM:* | Its a semantic "indexing" of the finalized alternatives to be implemented. |
| *DSA:* | It semantically bridges (non-finalized) design alternatives and summarizes design properties. |
| ***P7.*** | **How is the size of the design space documentation practically affected ?** |
| *UDM:* | It is increased by considering new user attribute values (i.e. polymorphism factor is increased). |
| *DSA:* | Practically, the design space is enlarged if more designers are engaged in the design process. |

Table 2: Key issues concerning the life-cycle of design alternatives in: (i) unified design method (UDM); and (ii) design space analysis technique (DSA).

On the other hand, the unified design method is primarily targeted on the production of finalized design artifacts into a form which is meaningful and mostly appropriate for the implementation of user-adapted interfaces. This is a fundamental difference between the two methods, which, as it will be explained later on, can be beneficially exploited through their combination into a comprehensive design procedure encompassing both process-oriented and artifact-

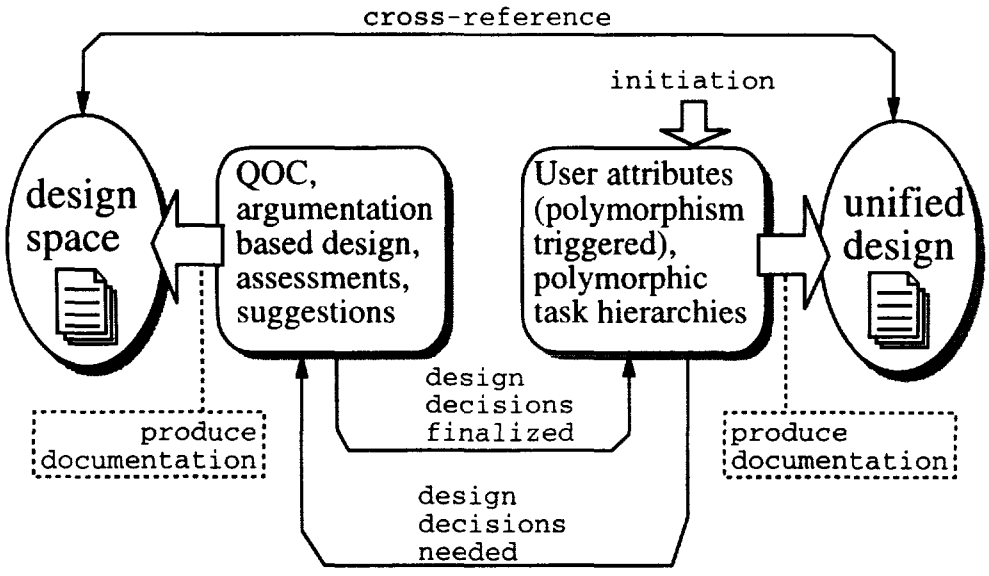


Figure 13: The fused design process model for combining the unified design technique with the design space analysis method.

oriented properties. Both the design space analysis and the unified design method generate design spaces consisting of alternative dialogue patterns. However, significant differences exist such as: the way design alternatives are generated, their relationship with respect to the final interface design, etc. We have identified seven key issues which concern the life-cycle of design alternatives within each design technique. We show how these issues are addressed by the unified design method (UDM) and the design space analysis method (DSA) in a manner demonstrating their artifact-oriented and the process-oriented nature respectively (see Table 2).

**Fusing unified design and design space analysis methods**

The design space analysis and the unified design method can be combined into a comprehensive design process. It is argued that any specialized design technique can benefit by being combined with the design space analysis approach. In our fused process model (see Figure 13), the overall design process is initiated by the polymorphic task analysis method. The "polymorphose" action of the unified design method will dictate the necessity for designing sub-hierarchies addressing specific user attribute values. In this context, such design decisions are analyzed and discussed by constructing appropriate design spaces. When final decisions are reached, they are fetched to the unified design sub-process, where they are recorded. Additionally, cross-references exist between the unified design documentation and the documentation of design spaces for: (i) elaborating design decisions (i.e. from the unified design documentation to design spaces' documentation); or (ii) summarizing design decisions (i.e. from design spaces' documentation to unified design documentation).

## SUMMARY AND CONCLUSIONS

The main purpose of the design information consolidated during the design process is to support subsequent development phases like target implementation and / or usability evaluation. In the context of user-adapted interaction, where an interactive application is able to adapt itself to individual end-user requirements, the design of alternative dialogue patterns is necessarily involved, reflecting the differing requirements and characteristics of end-users. Such alternative designed artifacts may exclude each other when designed for the same user tasks, if they are associated with incompatible user attribute values (e.g. expert / naive user, able / motor-impaired user).

The unified design method has been constructed in order to enable the expression of the user-adapted interface dialogue design artifacts into a single form. The polymorphic task structure, in combination with the explicit representation of user attributes and the expression of the selection logic for dialogue alternatives, is directly mapped to a modular component-based architectural model for user-adapted interfaces. The unified design approach, even though it explicitly introduces the notion of user representation and selection logic for alternative artifacts, it does not restrict the employment of any available user modelling tool or knowledge representation framework for design logic.

## ACKNOWLEDGEMENTS

The work reported has been partially funded by:

(i) The ACTS Programme of the Commission of the European Union (DG XIII), under the project AVANTI AC042 (*Adaptable and Adaptive Interaction in Multimedia Telecommunications Applications*). The partners of the AVANTI consortium are: ALCATEL Italia (Siette division);

IROE-CNR, Italy; Institute of Computer Science-FORTH, Greece; GMD, Germany; VTT, Finland; University of Sienna, Italy; TECO Systems, Italy; STUDIO ADR, Italy; MA Systems and Control, UK.

(ii) The TIDE Programme of the Commission of European Union (DG XIII), under the project ACCESS TP 1001 (*Development Platform for Unified Access to Enabling Environments*). The partners of the ACCESS consortium are: IROE-CNR, Italy; Institute of Computer Science-FORTH, Greece; University of Athens, Greece; RNIB, UK; University of Hertfordshire, UK; SELECO, Italy; MA Systems & Control, UK; Hereward College, UK; National R&D Centre for Welfare and Health, Finland; VTT, Finland; Pikomed, Finland.

**REFERENCES**

1. Akoumianakis, D., Savidis, A., Stephanidis, C. An Expert User Interface Design Assistant for Deriving Maximally Preferred Lexical Adaptability Rules. In *Proceedings of the 3rd World Congress on Expert Systems*, Seoul (Korea), 5-9 February 1996, 1298-1315.

2. Belloti, V. Integrating Theortecians' and Practicioners' Perspectives with Design Rationale. In proceedings of the *INTERCHI'93 conference on Human Factors in Computing Systems* (April 24-29), Amsterdam, Netherlands, 101-106.

3. Card, S., Moran, T., Newell, A. The phsychology of Human-Computer Interaction, Hillsdale, NJ, Lawrence Erlbaum.

4. Foley, J. D., Wallace, V. L., Chan, P.. The human factors of computer graphics interaction techniques. *IEEE Computer Gr. & Appl, 4, 11* (November 1984), 13-48.

5. Hartson, H. Rex., Siochi, Antonio. C., and Hix, Deborah. The UAN: A User-Oriented Representation for Direct Manipulation Interface Design. *ACM Trans. Inform. Syst.* 8, 3 (July 1990), 289-320.

6. Hill, R,D. Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction - The Sassafras UIMS. *ACM Trans. Gr. 5, 3* (July 1986), 179-210.

7. Hoare, C. A. R. Communicating Sequential Processes. In Commun. ACM 21, 8 (Aug, 1978), 666-677.

8. Johnson, P., Johnson, H., Waddington, P, Shouls, A. Task-related knwoeldge structures: analysis. modelling, and applications. In Jones, D. M.; Winder, R. (Eds), People and computers: from research to implementation, Cabridge University Press, 1988, 35-62.

9. Kobsa, A. Modelling the user's conceptual knowledge in BGP-MS, a user modelling shell system. Computational Intelligence 6, 1990, 193-208.

10. McLean, A., McKerlie, D. *Design Space Analysis and Use-Representations*. Technical Report EPC-1995-102, Rank Xerox, 1995.

11. Myers, B. A. A New Model for Handling Input. *ACM Trans. Inform. Syst.* 8, 3 (July 1990), 289-320.

12. Olsen, D. JR. Propositional Production Systems for Dialog Description. In *Proceedings of the CHI'90 Conference on Human Factors in Computing Systems* (April 1990), 57-63, ACM, New York.

13. Payne, S., J., Green, T. R. G. The user's perception of the interaction language: a two-level model. In Proceedings of the ACM CHI'83 Conference on Human Factors in Computing Systems, New York, 202-206.

14. Savidis, A., Stephanidis, C. Developing Dual Interfaces for Integrating Blind and Sighted Users: the HOMER UIMS. In proceedings of the *ACM CHI'95 conference in Human Factors in Computing Systems*, Denver, Colorado, May 7-11, 106-113.

15. Schneider-Hufschmidt, M., Kuhme, T., Malinowski, U. *Adaptive User Interfaces* (Eds). North Holland, 1993.

16. Stary, C. Integrating workflow representations into User Interface design representations. In Software Concepts and Tools, Vol. 17, December 1996.

17. Stephanidis, C. Towards User Interfaces for All: some critical issues. Panel session on *User Interfaces for All: Everybody, Everywhere, and Anytime*. In proceedings of the HCI International'95, Tokyo, Japan, July 9-14, Vol 1, 137-142.

18. Vergara, H. PROTUM - A Prolog basd tool for user modelling. Bericht Nr. 55/94 (WIS-Memo 10), University of Kostanz, Germany, 1994.

19. Wise, G. B., Glinert, E. P. Metawidgets for multimodal applications. In proceedings of the *RESNA'95 conference*, Vancouver, Canada, June 9-14, 455-457.