Chapter 21

The Unified User Interface Design Method

A. Savidis, D. Akoumianakis and C. Stephanidis

Abstract

This chapter describes the Unified User Interface design method. This new user interface design method has been developed to enable the "fusion" of different design alternatives, resulting from the consideration of differing end-user attributes 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 user interface developers. The Unified User Interface design method is elaborated here in terms of its primary objective, underlying process, techniques used, representation, and overall contributions to Human-Computer Interaction design.

1. Introduction

In its short history, Human-Computer Interaction (HCI) has accumulated a substantial body of knowledge which provides insight into the design of user interfaces. There is an abundance of design techniques which differ with regards to at least two dimensions, namely: (a) the underlying science base; and, (b) the type, range and scope of design outcomes, as well as the feedback they offer into the (re-) design process.

Regarding the science base, there are techniques, such the Hierarchical Task Decomposition (P. Johnson, H. Johnson, Waddington & Shouls, 1988), which are influenced by the Human Factors evaluation view of information processing psychology, and others, such as Goals, Operators, Methods and Selection Rules - GOMS (Card, Moran & Newell, 1983), Task-Action Grammar - TAG (Payne, 1984), the User-Action Notation - UAN (Hartson, Siochi, & Hix, 1990), etc., which adopt the methodological perspectives and assumptions of Cognitive Science. More recently, less formal techniques, such as inspection methods (Nielsen & Mack, 1994), have been introduced to provide more timely input and feedback to design activities.

Regarding design outcomes, the vast majority of existing techniques are artifact oriented. They do not explicitly record and document design rationale, thus, providing limited (if any) account of the rationale underpinning the various design options, or the decision making points that have shaped a particular design effort. More recently, there have been developments offering insights into the process, notations and tools for capturing, encoding and articulating design rationale (Moran & Carroll, 1996). These techniques focus explicitly on the process and argumentation through which design artifacts are generated.

Unified User Interfaces encapsulate automatically adapted behaviors and provide end-users with appropriately individualized interaction facilities. Hence, the process of designing Unified User Interfaces does not lead to a single design outcome (i.e., a particular design instance for a particular end-user). Rather, it collects and appropriately represents, alternative design "facets", as well as the conditions under which each of these should be instantiated (i.e., a kind of design rationale). Two major challenges that can be identified in this respect concern: (a) the process for the production of the various alternative designs; and, (b) the organization of all potential design instances into a single design structure.

Clearly, producing and enumerating distinct designs through the execution of multiple design processes is not a practical solution. Ideally, a single design process is desirable, leading to a design outcome that may directly be mapped to a single (i.e., unified) software system implementation. This, however, introduces two important requirements for a suitable design method. The first is that such a method should offer the capability to associate alternative artifacts (depicting different contexts of use) to a single design problem. The second requirement is that the method should preserve the *hierarchical discipline* of the HCI design process, by means of a "divide and conquer" strategy, in which design problems are incrementally broken-down and systematically addressed.

The first requirement leads to the definition of a *polymorphic design artifact*, as a collection of alternative solutions for a single design problem, where each alternative addresses different problem parameters. In this context, the problem can be in general identified as optimally designing artifacts for end-users and contexts of use, while the problem parameters are the various attributes characterizing the users, or the contexts of use. The second requirement points to the fact that polymorphic design artifacts should be hierarchically structured, thus giving rise to a *polymorphic task hierarchy*.

2. The Unified User Interface design method

On the grounds of the above, the *Unified User Interface design method* has been defined so as to address two objectives:

(i) enable the "fusion" of all potentially distinct design alternatives into a single unified form, without, however, requiring multiple design phases; and

(ii) produce a design structure which can be easily translated by user interface developers into an implementation form.

Some of the distinctive properties of this method are elaborated below by addressing its links with HCI design and by providing an overview of what the outcomes and design deliverables are.

2.1 Links with HCI design

The Unified User Interface design method is characterized by two properties which distinguish both the conduct of the method and its respective outcomes. First, the method adopts an analytical design perspective, in the sense that it requires an insight into how users perform tasks in existing task models, as well as what design alternatives and underpinning rationale should be embedded in the envisioned and re-engineered task models. In this context, the method links with other analytical perspectives into HCI design, such as design rationale and ethnography, to obtain the real world insight that is required, while it extends the traditional design inquiry by focusing explicitly on *polymorphism* as an aid to designing and implementing user- and use-adapted behaviors.

Secondly, the method supports a disciplined hierarchical approach to populating and articulating rationalized design spaces. This entails a middle-out design perspective, whereby enumerated design alternatives are fused into design abstractions and subsequently polymorphosed in a rational manner to facilitate automatic realization of alternative interactive behavior. In terms of conduct, the method is related to hierarchical task analysis, with the distinction that alternative 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 design polymorphism, which is characterized by the pluralism of plausible design options consolidated in the resulting task hierarchy.

Overall, the method 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 (Savidis, Paramythis, Akoumianakis & Stephanidis, 1997). The design process realizes an exhaustive hierarchical decomposition of various task categories, starting from the abstract level, by incrementally specializing in a polymorphic fashion (since different design alternatives are likely to be associated with differing 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) the recorded design rationale, for each design artifact produced, which has led to the introduction of this particular design artifact.

2.2 The design space

Design alternatives are necessitated by the different contexts of use and provide a global view of task execution. This is to say that design alternatives offer rich insight into how a particular task may be accomplished by different users in different contexts of use. Since users differ with regards 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.

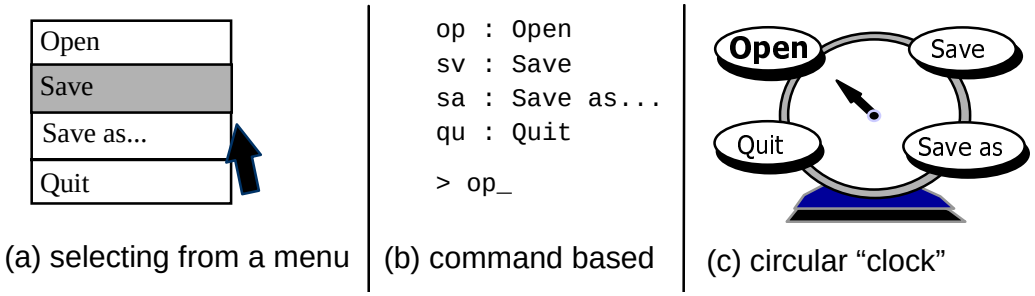

Figure 1: Alternative embodiments of selection in different design languages

As an example, consider the primitive interaction task of selection. A selection may be made either by choosing an option from a menu (see options (a) and (c) in Figure 1) or by issuing a command (option (b)), etc. Moreover, as illustrated by options (a) and (c) in Figure 1, the menu may be conveyed in different design languages[1]. For example, the use of the word “menu” in option (a) is borrowed from the “restaurant” domain of discourse; the command in option (b) follows the typewriter’s metaphor; the circular clock in option (c), resembles the operation of an electric device (e.g., a potentiometer). What is important to note, however, is that none of the above alternatives, or any other visual option that one may come up with, would be suitable for a blind user who lacks the capability to attain information conveyed in the visual modality. Instead, (physically or situationally) blind people would be more comfortable using alternative manifestations conveyed either through the audio or tactile modalities (see Figure 2).

[1] A design language is defined as a mechanism mediating the mapping of concepts in a source domain (e.g., restaurant, typewriter, electric device) to symbols in a presentation domain (e.g., interaction elements offered by a particular toolkit).

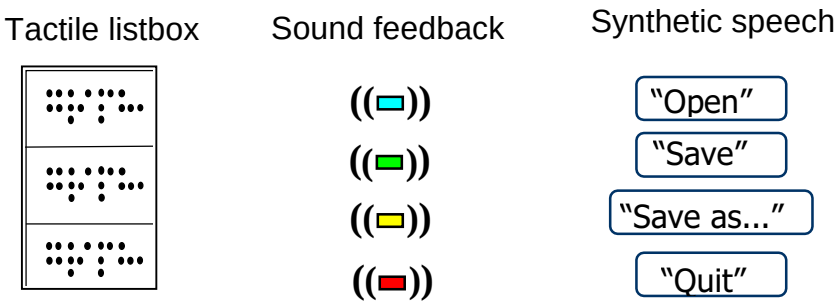

Figure 2: Non-visual alternatives for selection

2.3 Polymorphic task hierarchies

A polymorphic task hierarchy combines three fundamental properties: (a) *hierarchical organization*; (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 CSP (Communicating Sequential Processes) language for describing the behavior of reactive systems (Hoare, 1978), enable the expression of dialogue control flow formulae for task accomplishment. Figure 3 illustrates the basic set of operators provided; designers may freely employ additional operators as needed.

Operator	Explanation
before	*sequencing*
or	*parallelism*
xor	*exclusive completion*
*****	*simple repetition*
+	*absolute repetition*

Figure 3: Basic task operators in the Unified User Interface design method

The concept of polymorphic task hierarchies is illustrated in Figure 4. 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 polymorphic task hierarchy of Figure 4 shows how 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; and another realizing modal dialogue with 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 instantiating the selection dialogue (see Figure 4, lower-part): via scanning techniques for motor-impaired users, via 3D hand-pointing on 3D-auditory cues for blind people, via enclosing areas (e.g., irregular "rubber banding") for sighted users, and via 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 Figure 4 (lower part) graphical illustrations are used 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 & Hix, 1989).

The most common question regarding the need for a polymorphic task decomposition approach in the Unified User Interface design method challenges the argument that it is not sufficient to represent alternative task hierarchies by means of the traditional task-model, employing the *xor* operator among alternatives.

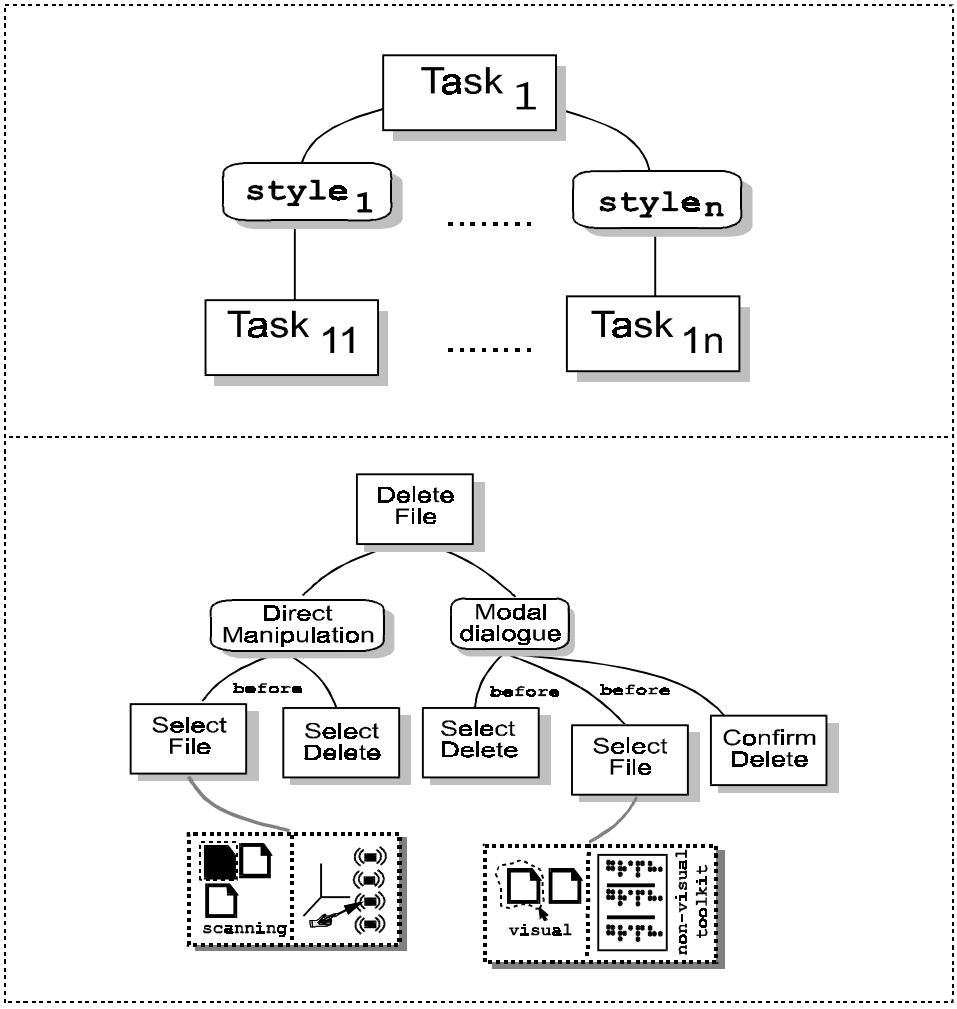

Figure 4: The polymorphic task hierarchy concept, where alternative decomposition "styles" are supported (upper part), and an exemplary polymorphic decomposition, for two different user groups, namely blind and motor-impaired users (lower part)

The answer to this question requires some elaboration (see also Figure 5). Firstly, the *xor* operator in the traditional task-model is interpreted as "the user is allowed to perform any, but only one, of the *N* sub-tasks". This means that the physical interaction context (i.e., interface components) for performing any of the sub-tasks is made available to the user, while the user is required to accomplish only one of those sub-tasks. However, if alternative sub-hierarchies are related via polymorphism, it is implied that "a particular end-user will be provided with the design (out of the N alternative ones) that maximally matches the specific user's characteristics". Clearly, it is meaningless to provide all designed artifacts (which are likely to address diverse user characteristics) concurrently to end-users, and force users to

work with only one of those. Hence, the *xor* operator is not the appropriate way for organizing alternative dialogue patterns.

As discussed in more detail later on, 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.

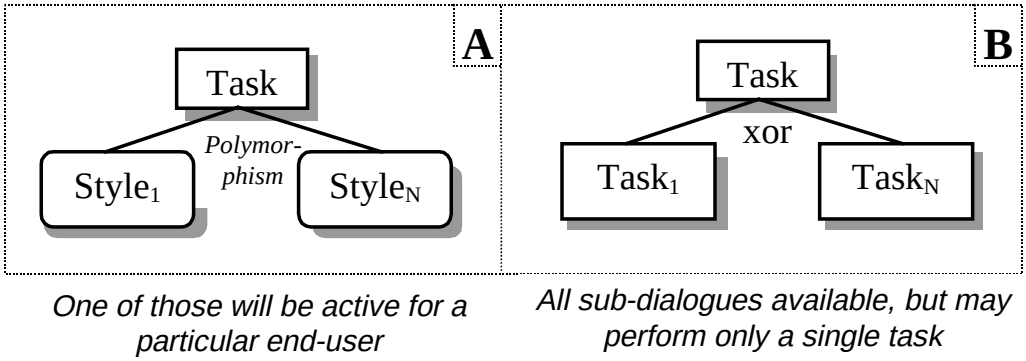

Figure 5: (a) The use of polymorphism for alternative styles (dialogue patterns), when those address different user- and usage-context- attributes; and (b) the xor operator for various tasks, when exclusive completion has to be imposed

2.4 Adaptation rationale

When a particular task is subject to polymorphism, alternative sub-hierarchies are designed, each 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. If polymorphism is not applied, a task model merely represents a single interface design instance, on which further run-time adaptation is restricted; in other words, *there is a fundamental link between adaptation capability and polymorphic 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. We will use the physical design artifact of Figure 6, 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 part of the task-based design, in two ways:

(a) through polymorphism, where each of the two dialogue artifacts is defined as a distinct style; the two resulting styles are defined as being *compatible*, which implies that they may co-exist at run-time (i.e., the end-user may freely use the command line or the interactive file manager interchangeably);

(b) via decomposition, 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.

These two alternative approaches are illustrated in Figure 7.

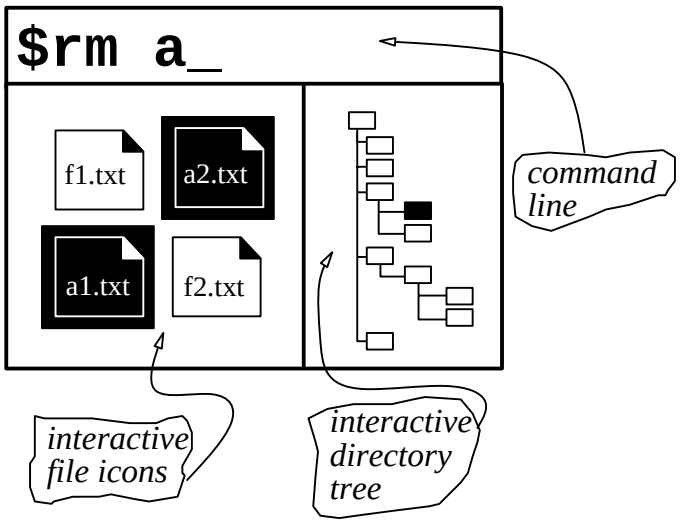

Figure 6: A design scenario for alternative concurrent sub-dialogues, in order to perform a single task (i.e., task multimodality)

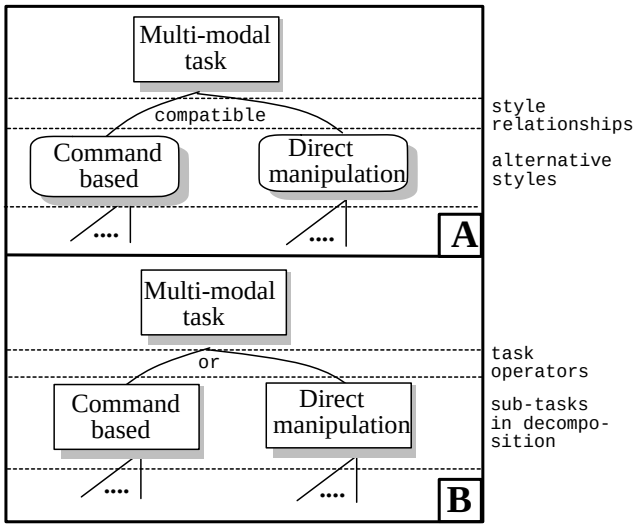

Figure 7: Two ways for representing design alternatives when designing for task-level multimodality: (A) via polymorphism, adding run-time control on pattern activation; and (B) via the *or* operator, hard-coding the two alternatives in a single task implementation

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, 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).

3. Conducting polymorphic task decomposition

In this section, we provide a consolidated account of how the Unified Interface design method can be practiced.

3.1 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 (see Figure 8):

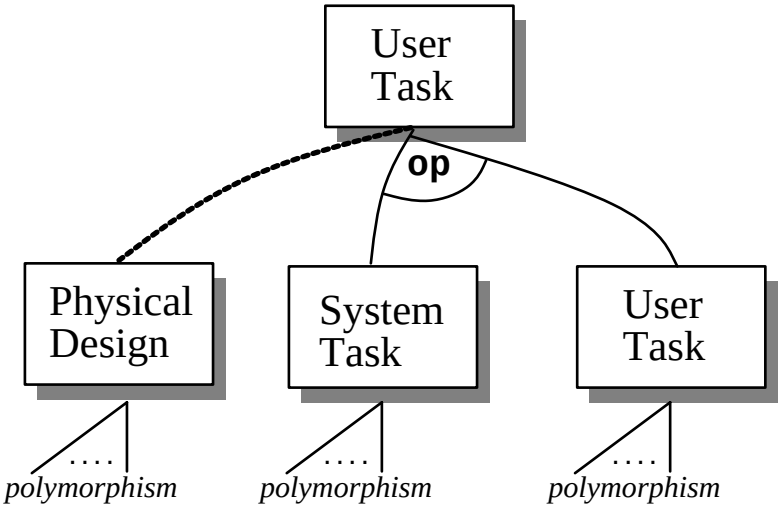

Figure 8: The three artifact categories in the Unified User Interface design method, for which polymorphism may be applied, and how they relate to each other

(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, they are treated in the same manner as user tasks.

(c) *Physical design*, which concerns the various interface components on which user actions are to be performed; 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 task. It provides the physical dialogue pattern associated to a task-structure definition. Hence, it simply plays the role of annotating the task hierarchy with physical design information. An example of such annotation is shown in Figure 4, 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, on the basis of 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 (see also Chapter 3 “International and Intercultural User Interfaces” in this volume).

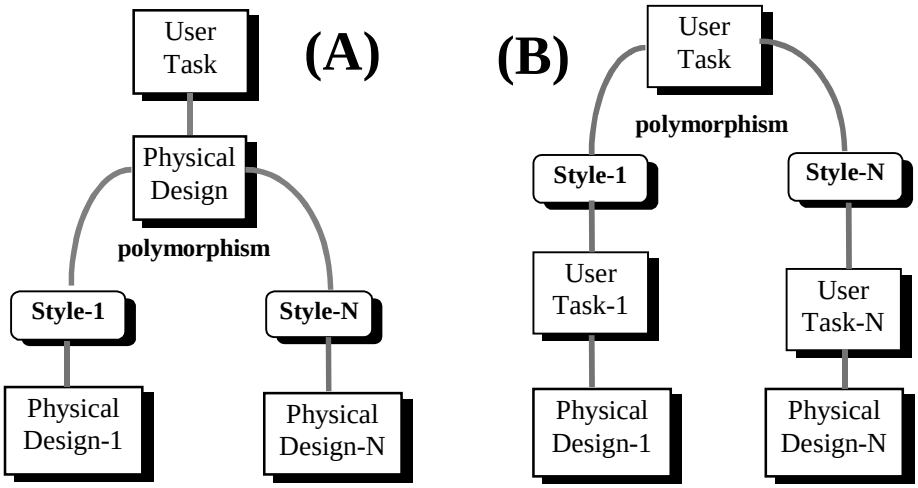

Figure 9: Representation of alternative physical artifacts: (A) in the case of the same non-polymorphic task; and (B) in the case where polymorphic task decomposition is needed

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 polymorphic artifacts is: *if alternative 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 artifact* (see Figure 9, part A). *If alternative designs are needed due to alternative task structures (i.e., task-level polymorphism), then each alternative physical design should be assigned to its respective style* (see Figure 9, part B).

3.2 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 “selection”, then, clearly, the physical means of performing the selection 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 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 lies 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 illustrated in Figure 10; the decomposition starts from abstract- or physical- task design, depending on whether 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 Figure 10.

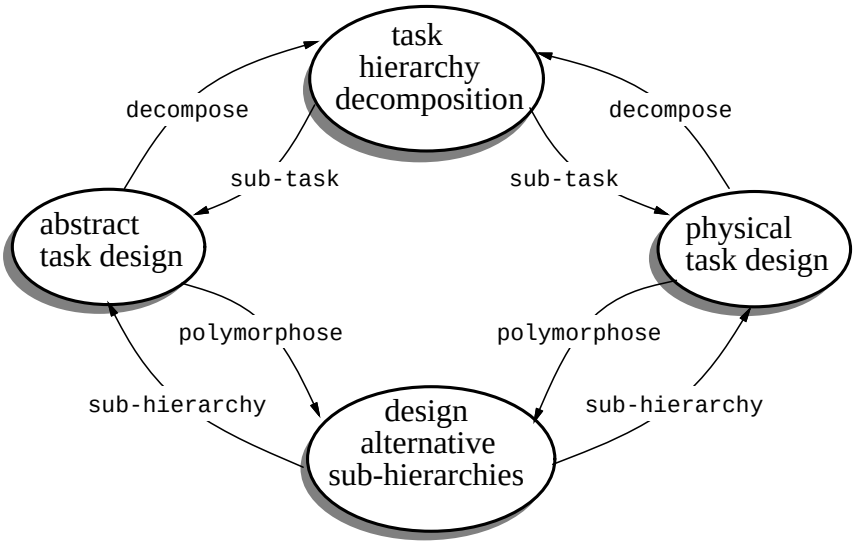

Figure 10: The polymorphic task decomposition process in the Unified User Interface design method

3.2.1 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 via a *decomposition* action, leading to the *task hierarchy decomposition* state. In the case of a polymorphic decomposition, the transition is realized via a *polymorphose* action, leading to the *design alternative sub-hierarchies* state.

3.2.2 Transitions from the "design alternative sub-hierarchies" 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.

3.2.3 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.

3.2.4 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.

3.3 An example of polymorphic task decomposition

To illustrate the process of polymorphic task decomposition (see Figure 10), we will refer to an example which is depicted in Figure 4 (lower part). The sequence of steps is illustrated in Figure 11 (states are mentioned with brief names).

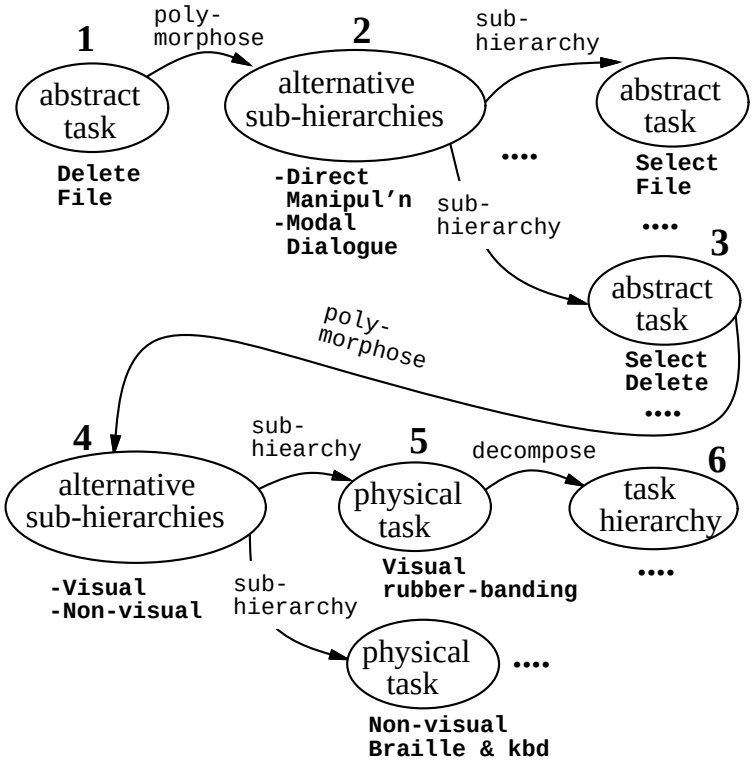

Figure 11: An example of a polymorphic task decomposition process diagram

The initial state is *abstract task* (step 1), since "Delete File" can be defined as an abstract task. Through the *polymorphose* transition, two alternative styles are defined, resulting in two distinct sub-hierarchies (step 2). Each such sub-hierarchy is further decomposed, by firstly deciding whether the top-level task is an abstract or a physical task, so as to continue the process (step 3). For instance, both the "Select File" and the "Select Delete" tasks are abstract (the rest of the tasks are not shown for clarity). Then, the steps for the "Select Delete" task are shown; this task is polymorphosed (step 4), resulting in two alternative sub-hierarchies (one for 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 subsequently decomposed (step 6) to a unimorphic task hierarchy. It should be noted that, instead of pursuing the polymorphic task decomposition approach, we could alternatively continue from step 6, via any other appropriate design practice, such as for example, event modeling.

3.4 Designing alternative styles

The polymorphic task model provides the design structure for organizing the various alternative dialogue patterns of automatically adapted interfaces into a unified form. Such a hierarchical structure realizes the fusion of all potential distinct designs which may be explicitly enumerated given a particular Unified User Interface. 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-rationale, connecting the designed styles with particular user- and usage-context- attribute values, is documented.

3.4.1 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 Figure 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*. An "accessibility gap" is usually encountered when there is a particular decomposition (for user- or system- tasks, as well as for physical design) which does not address some combination(s) of the decision attribute values. Such a design gap can be remedied by constructing the necessary alternative sub-hierarchy (-ies) addressing the excluded decision attribute values.

3.4.2 Constructing the space of decision parameters

We will discuss the definition of user attributes, which are of primary importance, while the construction of context attributes may follow the same representation approach. In the Unified User Interface design method, end-user representations may be developed using any suitable formalism which can encapsulate user characteristics in terms of attribute-value pairs. 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.

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**	

Figure 12: An example of a user-profile, as a collection of values from the value domains of user-attributes, from (Savidis, Akoumianakis, & Stephanidis, 1997)

The value domains for each attribute class are chosen as 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 various individual end-users. For instance, commercial systems realizing a single design for an "average" user have no differentiation capability at all. 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 Figure 10. For simplicity, designers may choose to elicit only those attributes from which differentiated design decisions are likely to emerge.

3.4.3 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 various dialogue artifacts which 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").

Exclusion	*Relates many styles*. Only one from the alternative styles may be present.
Compatibility	*Relates many styles*. Any of the alternative styles may be present.
Substitution	*Relates two groups of styles together*. When the second is made "active" at run-time, the first should be "deactivated".
Augmentation	*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".

Figure 13: Design relationships among alternative styles, and their run-time interpretation

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 Figure 6 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 (see Figure 13), defining whether alternative styles may be concurrently present at run-time. We will now show how these four fundamental relationships reflect pragmatic, real-world design scenarios.

Exclusion

The exclusion relationship is applied when the various alternative styles are deemed to be usable only within the space of their target user- and usage-context- attribute values. For instance, assume that 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 in which 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 (see Figure 6 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 on the basis of monitoring data collected during interaction, through which information such as frequency of use and repeated usage patterns can be extracted. Hence, particular physical design styles would need to be "cancelled", 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 “cancelled”, being substituted by other styles which 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 us 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.

3.5 Engaging abstract interaction objects

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 all 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, reflecting different user-, usage-context-, and domain-properties.

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 behaviors and objects. In this context, we have defined a role-based model (see Figure 14) 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, namely lexical, syntactic and semantic; the description of each role follows.

3.5.1 Lexical role

In this 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). The content could be verbal (i.e., the string is a phrase), if the user understands natural language, or 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, or other visual / auditory effects) concern the physical implementation that may have alternative realizations.

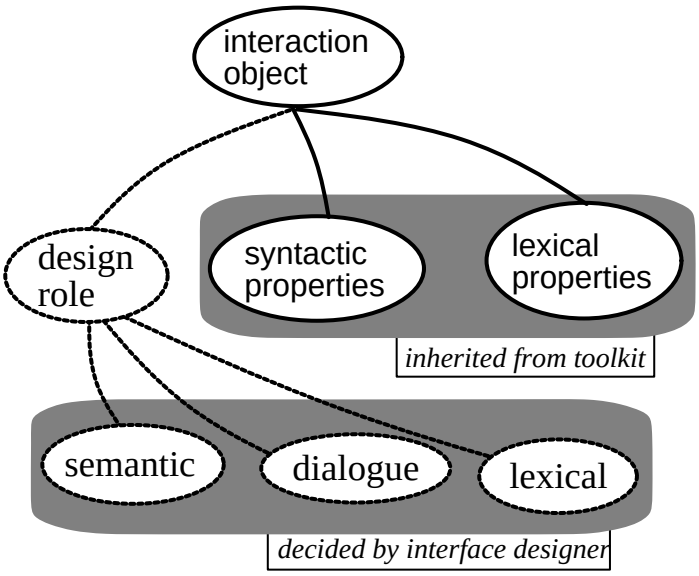

Figure 14: Role-based model of interaction objects

3.5.2 Syntactic role

The interaction 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, all play the role of a "Command" given by the user, in the particular dialogue context. Such a "Command" class may be used to support, for instance, execution, confirmation, or cancellation tasks, and may be applicable under various interaction metaphors. It could be physically realized as a conventional push-button for the desktop windowing interaction metaphor, as a voice-input command object for non-visual interaction, and as a particular symbol structure for language-impaired users. The abstract interaction object "Command" may have only one Boolean attribute to control, whether it is accessible or not, while the presentation feedback for indicating accessibility status could be different, depending on its physical realization.

3.5.3 Semantic role

In this case, an interaction object interactively realizes a domain object. For instance, an interaction object may present a domain object's 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 into 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).

3.6 Re-engineering designs through the role-based model

The role-based model can be applied on an existing physical design, in order to produce a higher-level design scenario. Such a scenario will serve as an abstract design representation, which may form the basis for deriving further alternative physical design scenarios. This notion of "filtering" physical scenarios via the role-based model, so as to subsequently construct a higher-level design representation, is illustrated in Figure 15. We will demonstrate the power of such a design re-engineering process through an example.

Figure 16 depicts a form-based dialogue, typically found in Web documents, for providing credit card information. In Figure 17, the physical design scenario is analyzed in order to identify object roles. The resulting higher-object model is depicted in the diagram of

Figure 18. Figure 19 and Figure 20 illustrate two alternative physical realizations which can be derived and which comply with the abstract scenario of Figure 17. The realization depicted in Figure 18 indicates one potential option for graphical interaction, while the corresponding depiction of Figure 19 presents a non-visual scenario which conveys the same information in an alternative metaphoric representation, namely that of the room.

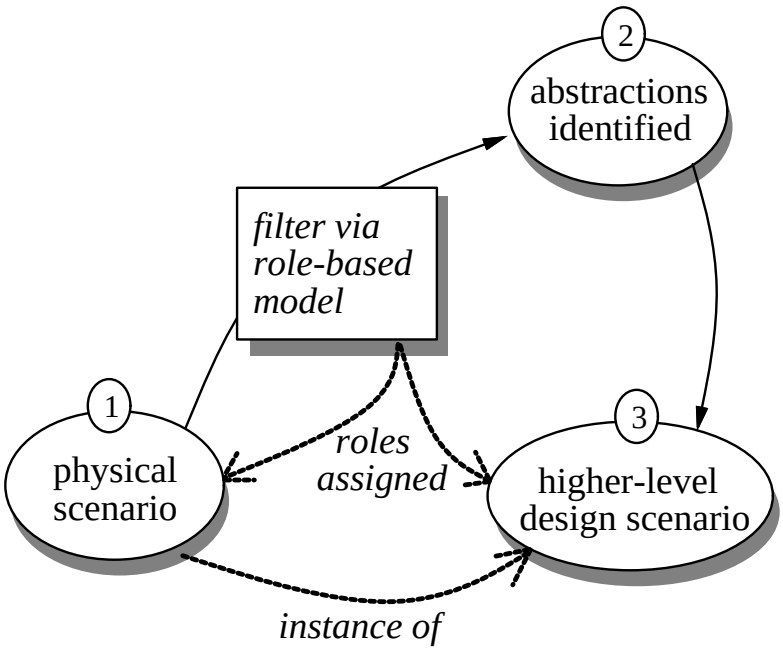

Figure 15: Producing higher-level design scenarios through the role-based model

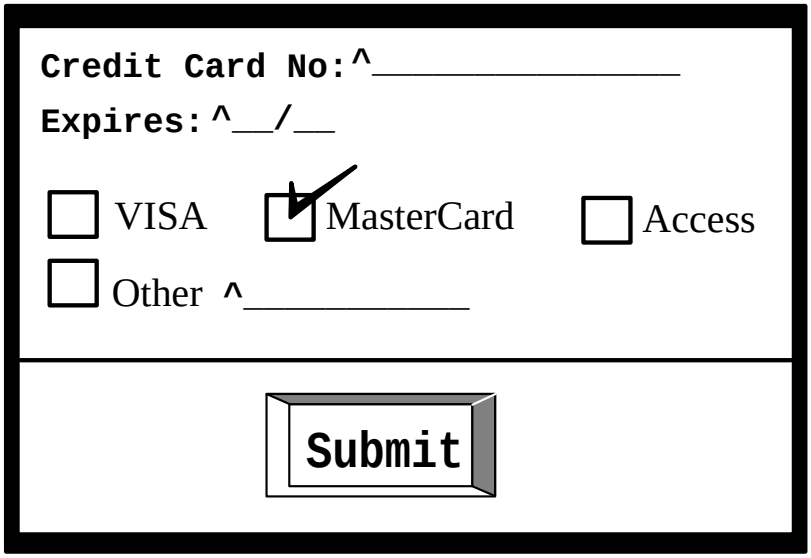

Figure 16: The physical design scenario which will be re-engineered

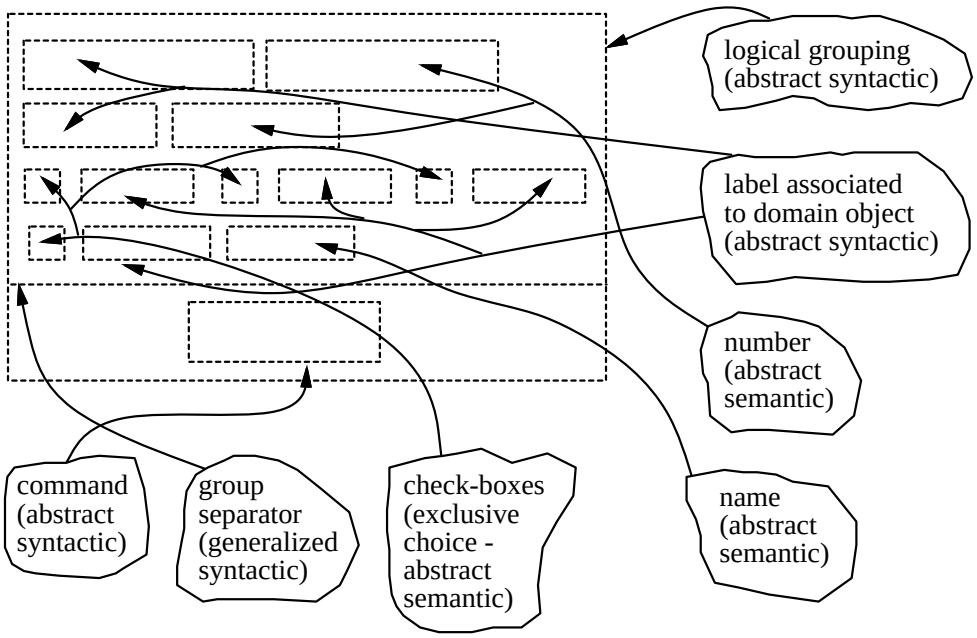

Figure 17: Assigning roles to physical interaction objects

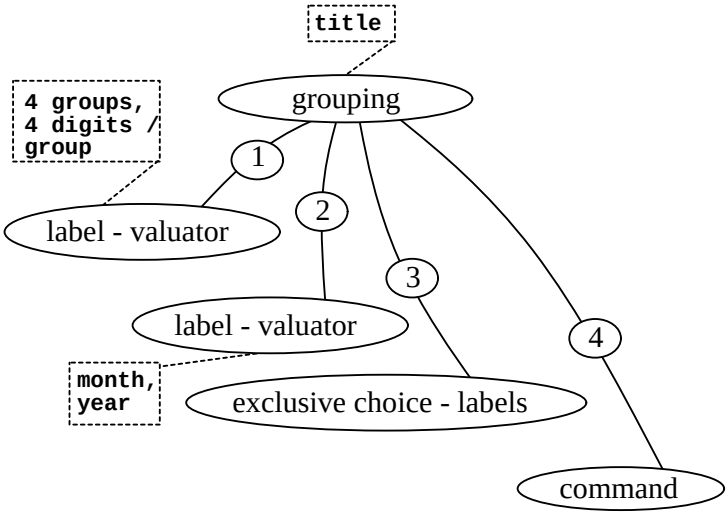

Figure 18: The resulting higher-level object model

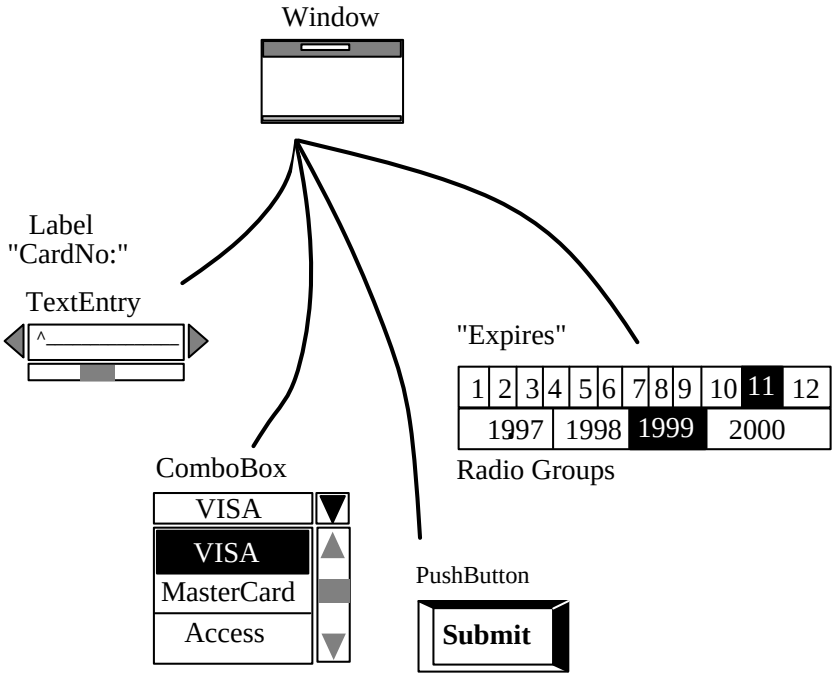

Figure 19: An alternative graphical design derived on the basis of the abstract object model

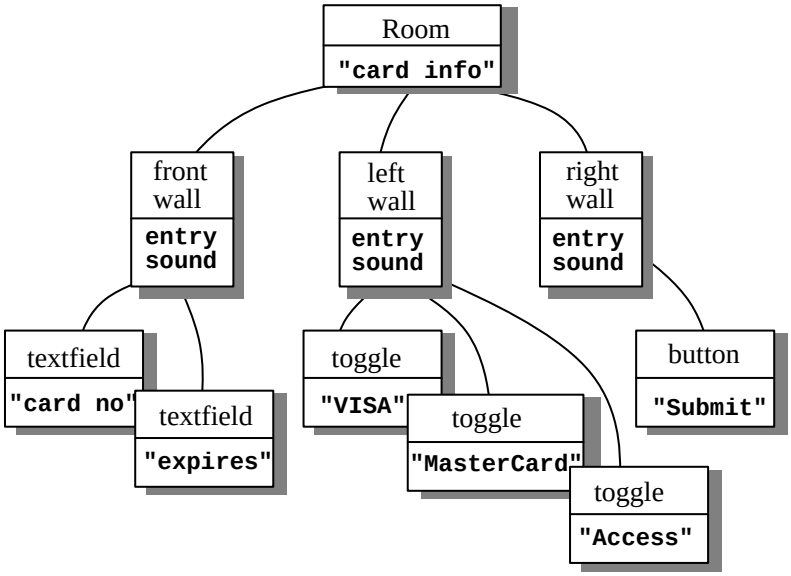

Figure 20: An alternative non-visual Rooms design, derived from the higher-level object model

4. Discussion and conclusions

This chapter 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, 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 to be especially relevant for the design of systems which are required to exhibit adaptable and adaptive behavior, in order to support individualization to different target user

groups. 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.

The main conclusion from this chapter is that interaction design becomes increasingly a knowledge-intensive endeavor. Designers should, therefore, 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. To this end, analytical design methods, such as Unified User Interface design, will become necessary tools for capturing and representing the global execution context of interactive products and services in the emerging Information Society. 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.

References

Card, S.K., Moran, P.T., & Newell, A. (1983). *The psychology of Human Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

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., Siochi, 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 Sassafras UIMS. *ACM Transactions on Graphics*, 5 (3), 289-320.

Hoare, C.A.R. (1978). Communicating Sequential Processes. *Communications 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 D.M. Jones, & R. Winder (Eds.), *People and computers: from research to implementation - Proceedings of HCI '88* (pp. 35-62). Cambridge University Press.

Marcus, A. (1996). Icon Design and Symbol Design Issues for Graphical Interfaces. In E. Del Galdo, & J. Nielsen (Eds.), *International User Interfaces* (pp. 257-270). New York: John Wiley and Sons.

Moran, T.P. & Carroll, J.M. (1996). *Design Rationale: Concepts, Techniques, and Use*. Hillsdale, NJ: Lawrence Erlbaum Associates

Nielsen, J. & Mack, R.L. (Eds.) (1994). *Usability Inspection Methods*. New York: John Wiley & Sons.

Payne, S. (1984). Task-action grammars. In *Proceedings of IFIP Conference on Human-Computer Interaction: INTERACT '84 (Vol. 1)*, London, England (pp. 139-144). Amsterdam: North-Holland, Elsevier Science.

Saldarini, R. (1989). Analysis and Design of Business Information Systems. *Structured Systems Analysis* (pp. 22-23). New York: MacMillan Publishing.

Savidis, A., Akoumianakis, D., & Stephanidis, C. (1997). Software Architectures for Transformable Interface Implementations: Building User-Adapted Interactions. In *Proceedings of HCI International '97*, San Francisco, California (pp. 453-456). Amsterdam: Elsevier, Elsevier Science.

Savidis, A., Paramythis, A., Akoumianakis, D., & Stephanidis, C. (1997). Designing user-adapted interfaces: the unified design method for transformable interactions. In *Proceedings of the ACM conference in Designing Interactive Systems (DIS '97)*, Amsterdam, The Netherlands (pp. 323-334). New York: ACM Press.