Encapsulating intelligent interactive behaviour in unified user interface artefacts

D. Akoumianakis, A. Savidis, C. Stephanidis*

Institute of Computer Science (ICS), Foundation for Research and Technology-Hellas (FORTH), Science and Technology Park of Crete, Heraklion, Crete, GR-71110, Greece

Abstract

Intelligence at the level of the user interface is currently being supported through a number of prevalent strands, including adaptive user interfaces, model-based user interface development and interface agents. Moreover, the term intelligent user interface typically implies the notion of dynamically enhancing the interaction with a single implemented artefact to suit different usage patterns, user groups, or contexts of use. This article extends this notion and describes how unified design artefacts can support the development of accessible and high quality user interfaces exhibiting the characteristics of multiple metaphor environments. To this effect, the article outlines the principles of unified user interface development and discusses how it can be used to advance Intelligent Interface Technology to account for diverse user requirements and interaction contexts. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Intelligent user interfaces; Multiple metaphor environment; Unified user interface development method

1. Introduction

The need for intelligent and co-operative Human–Computer Interaction (HCI) is driven by several contemporary technological developments, including: the increasing complexity of human–machine dialogues [1]; the increasingly knowledge-based nature of tasks resulting from the radical changes in both the nature of work and the content of tasks [1]; the increasing number of computer users characterised by diverse abilities, skills, requirements and preferences; the wide proliferation of technologies incorporating advanced interaction facilities, novel input/output devices and multimodal interaction techniques based on advanced multimedia interaction technologies. These contemporary developments

* Corresponding author. Tel: +30-81-391741, fax: + 30-81-391740, URL: http://www.ics.forth.gr/proj/at-hci/cs.html.

E-mail address: cs@ics.forth.gr (C. Stephanidis).
raise a compelling need towards a comprehensive framework for designing, implementing and evaluating user interface adaptations.

Traditionally, the adaptive behaviour was largely attributed to the human partner, as systems did not embody the required sophistication to identify the need for, and to subsequently implement adaptive control. As a result, the human operator was required to compensate for the shortcomings of the machine and to supply the capabilities necessary to accomplish the task. However, recent developments in the fields of Artificial Intelligence and their application to User Interface Software and Technology (UIST) problems, have enabled a greater degree of co-operativity in human–computer dialogues, as well as the implementation of inferential reasoning towards adaptive control on the part of the machine. User interfaces exhibiting such characteristics are frequently referred to as intelligent user interfaces.

In recent years, the field of Intelligent Interface Technology has experienced substantial progress and growth [2]. This becomes evident from the increasing popularity of the topic amongst academic/research and industrial practice. It is expected that this trend will continue as existing technologies mature and provide the driving concepts towards a new techno-paradigm shift, which characterises the emergence of the Information Society. Although Intelligent Interface Technology still poses numerous challenges and the contributing fields are still attempting to ascertain concepts and match them with practice (e.g. current debates on model-based interface development, intelligent agents vs. direct manipulation), several of their premises are highly valued and relevant in the context of the emerging Information Society. A review of the relevant literature identifies some of these premises and motivating principles, such as individualisation, co-operativity, usability, task simplification, etc., and provides numerous prototypical implementations of interactive systems exhibiting the characteristic properties of Intelligent Interface Technology. However, there is yet another cluster of design issues, which, though in the periphery of the tradition of Intelligent Interface Technology, have become increasingly relevant in the light of the communication and interaction-intensive contexts of use in the emerging Information Society; these relate to the concepts of accessibility and interaction quality.

The issue of accessibility concerns the right of all citizens to obtain and maintain access to a society-wide pool of information resources and artefacts, given the varieties of context. To this effect, accessibility implies an explicit design focus to consider the relevant issues, as opposed to an afterthought. Interaction quality implies quality in the use of these information resources and artefacts by humans in the various problem-solving, information seeking and communication-intensive activities. This notion of quality goes beyond the “traditional” notion of usability (i.e. measurable aspects based on performance criteria such as effectiveness, efficiency, satisfaction, etc), to include aspects (such as usefulness, suitability for task, tailorability, etc.), which may not be measurable with currently available means.

The compelling need to address these two requirements was revealed in the context of European Commission funded collaborative research and development work (see Acknowledgements) aiming to provide and demonstrate the feasibility of tools for building user-adapted interfaces accessible by different user groups, including disabled and elderly people [3,4].

To this end, Intelligent Interface Technology is relevant, but the currently available
wisdom in the field seems inadequate to address the peculiarities of the domain of disability. In particular, Intelligent Interface Technology has traditionally not addressed accessibility and interaction quality in the context and scope considered in the present work. As a result, there is a genuine lack of sufficiently consolidated knowledge and reference examples to inform and guide practitioners. Such a body of knowledge is needed to identify and characterise the type of intelligent systems behaviour that needs to be supported so as to enable and facilitate access, as well as present the architectural abstractions that preserve the required characteristics. To this effect, the article reviews the state-of-the-art in Intelligent Interface Technology with the aim to identify some of the challenges posed when considering accessibility and interaction quality, and subsequently, discusses a particular approach to the design of systems that are enabling, co-operative, user- and use-adapted. By this account, the article seeks to shed light into the realities of Intelligent Interface Technology in the domain of disability, as well as to advance a proposition for constructing user interfaces accessible and usable by the broadest possible end-user population.

The article is structured as follows: The next section briefly reviews the prevailing strands of Intelligent Interface Technology, discusses their underpinning assumptions and identifies shortcomings, or challenges to be addressed. Subsequently, the notions of multiple metaphor environments and context-sensitive processing are introduced and discussed in the light of the identified challenges. Then, the unified user interface development method is presented in terms of key properties and methodological contributions towards the construction of multiple metaphor environments suitable for diverse user groups, including people with disabilities. The next section reports on some of the technical features of a supporting tool environment for unified user interface development and discusses practice and experience with the method. Finally, the article discusses the implications of unified user interface development on popular HCI practices and highlights the potential contributions of this work to the study of Intelligent Interface Technology and its realities. The article concludes with a summary and directions for future work.

2. Related work

Over the years, research on intelligent user interfaces has followed a multitude of alternative strands (for a review see [2]), building upon notions such as natural language understanding, explanation systems, intelligent tutoring systems, intelligent help systems, computer-aided instruction, multi-modal systems, model-based development, intelligent presentation systems, and agent-based interaction. As a result, several paradigms have been established each advocating a particular perspective upon the design, implementation and evaluation of intelligent interaction. The most prevalent paradigms include user interface adaptation, model-based user interface development and interface agents.

2.1. User interface adaptation

Adaptation in user interface software is typically served by either adaptivity and/or adaptability. An adaptive user interface changes dynamically in response to its experience
with users [5]. This is typically achieved by dedicated software tools (e.g. user modelling components) responsible for: (i) acquiring and maintaining user models, and (ii) reasoning towards suitable user interface adaptations. Adaptive user interfaces build upon advances in intelligent tutoring systems, student and user modelling, explanation systems and knowledge representation. A review of the state of the art in adaptive user interfaces can be found in [5,6], while they have been the focal point of concern in a wide variety of research efforts [7–9]. Moreover, the recent literature reports several software tools which have been developed to support adaptive interaction [10–14].

An adaptable user interface, however, provides tools, which allow the user to tailor certain aspects of the interaction, while using a system. Some of the early attempts to construct adaptable systems are OBJECTLENS [15], BUTTONS [16], Xbuttons [17]. Adaptability has also been the chief objective in the development of the PODIUM system [18]. All these systems allow the user to modify certain aspects of their interactive behaviour while working with them. More recently, the AURA project (Adaptable User Interfaces for Reusable Applications) of the ESPRIT-II Programme of the European Commission has investigated thoroughly the issue of adaptability [19] and the underlying architectural abstractions for adaptable systems.

User interface adaptations build upon research on user and discourse modelling and provide the field of Intelligent Interface Technology with the required tools to address the objectives of customisation, individualisation and tailorability. However, comprehensive evaluation instruments and techniques for guiding the design of adaptations are still missing. This is primarily due to the contextual nature of adaptations, which do not facilitate objective assessments.

2.2. Model-based interface development

Model-based interface development involves the use and articulation of reusable models and knowledge repositories encapsulating the wide variety of details pertaining to user interface development. A comprehensive review of recent work and future challenges for this paradigm can be found in [20]. Exemplar systems and research prototypes complying to the requirements of model-based interface development include UIDE [9], HUMANOID [21], TRIDENT [22] and MASTERMIND [23]. In general, model-based development offers several contributions to Intelligent Interface Technology. First of all, model-based development fosters a shift towards declarative specifications of interfaces. Secondly, it promises to decrease both the time and expertise required to create interfaces, through reusable models, automation and design support tools. Thirdly, model-based development enables novel forms of designer support such as critiquing, design refinements and incremental updates.

However, the current generation of model-based interface development tools has not appropriated the full range of the above potentialities. In fact, a frequently cited critique is that model-based development is still far from exercising the impact on user interface software technologies that was originally anticipated.

2.3. Agent-based interaction

This paradigm entails the use of software agents to delegate responsibility regarding
various aspects of interaction. Intelligent agents are entities capable of voluntary, rational action carried out in order to achieve certain goals [6]. They usually hold a representation of belief in the state of the world and are equipped with facilities to learn from patterns of behaviour, from a single user, or other users, or even other agents. Interface agents have built upon technical advances such as the development of knowledge interchange and agent communication languages, such as KQML [24] which enables human–agent and agent–agent communication and co-ordination, as well as advances in agent competencies which, coupled with advanced interface components (e.g. speech, facial animation, etc), make anthropomorphic agent emulation feasible. For a comprehensive review of agent-based interaction, implemented systems and design guidelines, see [2,25].

Agents offer several new insights to the study of intelligent user interfaces. These are grounded on the type, range and scope of mediation that agent technology advocates and the potential implications on the user’s experience. Specifically, agents foster a communication- and collaboration-oriented paradigm for computer-mediated human activities, which promises to decrease human workload and make the overall experience of interaction less stressful and more productive.

2.4. Prevailing assumptions and shortcomings

In the context of Intelligent Interface Technology, the above prevalent paradigms share a common denominator. To illustrate this, let us consider the example of a visual desktop menu, or a toolbar intended to convey choice amongst related items. An intelligent user interface may facilitate the user’s selection through either adaptive ordering of options (based on frequency of use data), or adaptive prompting (based on contextual information resulting from a model), or guidance (with the help of an agent). In each case, the intelligent user interface attempts to enhance the interaction with the same physical artefact1, namely the desktop menu, or the toolbar, though using different strategies. This implies a notion of intelligence as entailing dynamic enhancements of the interaction with a single implemented artefact. Such an assumption raises several implications on the type and range of intelligent behaviour that an interface exhibits.

Referring back to our previous example of adaptive selection, let us assume that the same system is to be also used by a blind user who cannot initiate and sustain interaction due to the lack of physical and/or cognitive capabilities to articulate a visual modality. A truly intelligent interface should be able to anticipate such limitation and undertake appropriate actions towards a new state that would facilitate the blind user’s interaction requirements. However, it is evident that the strategies identified above (e.g. adaptive ordering, adaptive prompting and guidance) do not provide the interface with the ability to anticipate the situation of a blind user. Consequently, the more general questions that arise are how the above strategies for enhancing the user interaction would suit users with diverse abilities and requirements and how today’s intelligent user interfaces can cope with such radical departures from a predetermined interaction paradigm, such as the visual desktop metaphor.

1 By physical artefact, we imply an interaction element (e.g. an interaction object or a composite component, such as a dialogue) bound to a particular toolkit (e.g. Windows95™, Motif).
A potential solution would be to revise our initial assumption that intelligence entails dynamically enhancing the interaction with a single design artefact, into one that treats intelligence as context-sensitive processing, which encompasses:

1. the identification of plausible alternatives (such as the desktop menu, the 3D acoustic sphere for the blind [26]) in anticipation of the variety in the context of use that the artefact is to be encountered;
2. the unification of alternative concrete design artefacts into abstract design patterns, or unified design artefacts (such a generalised container);
3. a method to allow assessment of alternatives in relation to the current objective and the mapping of an abstract design pattern into the appropriate concrete/physical artefact; and,
4. the capability to dynamically enhance interaction with each one of these multiple physical artefacts, as the need arises.

A user interface complying to the above notion of intelligence would be able to dynamically undertake the required transformation, prior to and during interaction, so as to provide the appropriate interactive behaviour for each user category to accomplish a given task. In this sense, intelligent interactive systems are considered as multiple metaphor environments capable of performing context-sensitive selection amongst plausible alternatives [27].

3. Multiple metaphor environments and context sensitive processing

Before we describe the notion of a multiple metaphor environment and its characteristic properties, it is perhaps important to briefly present our understanding of a metaphor and its role within the user interface software architecture. The study of metaphors for effective interface design is not new. It has been addressed in the past by several researchers, from various perspectives, including methodologies for studying metaphors [28], embedding metaphors into user interface implementations [29,30], graphics design [31], etc. However, despite these efforts, very little is known as to how different metaphors can be embedded into computer-based interactive software.

In the context of the present work, metaphors are considered to have a two-fold purpose in interface design. They can either be embedded in the user interface, or characterise the overall interactive environment of an application. For example, the menu interaction object class, as commonly encountered in popular user interface development toolkits, follows the “restaurant” metaphor, and provides an example of embedding metaphor into a user interface. This is because it is commonly found as embedded element in systems conveying radically different interactive embodiments of the computer; examples are the visual desktop as in Windows95™, rooms as in [29], or book as in [30].

Alternatively, a metaphor may characterise the properties and the attitude of the overall interaction environment. For instance, the visual embodiment of the desktop metaphor in Windows95™ presents the user with an interaction environment based on high level containers, such as sheets of paper called windows, folders, etc., which characterise the overall interactive embodiment of the computer. Systems such as those in [29] or [30], are
examples of alternative embodiments of real world metaphors into a user interface. It
should be noted that, a particular real world metaphor may have different interactive
instantiations. Thus, for example, OSF/MOTIF™ and Windows95™ support variations
(mainly in the look and feel) of the visual embodiment of the desktop metaphor. From the
above, it follows that the interactive environment of a metaphor is realised by specific user
interface development toolkits.

Different interaction metaphors may be facilitated either through the enhancement, or
augmentation of existing development toolkits, or by developing new toolkits. For
instance, an enhancement of the interactive environment of a metaphor may be facilitated
by introducing new composite object classes, such as the note cards in prevailing
Windows-like systems, or by embedding in the toolkit additional interaction techniques,
such as automatic scanning facilities for interaction object classes [32]. What is important
to note about enhancement, or augmentation is that it rarely alters the overall interactive
environment of the metaphor. This is because the scope of the enhancement or augmenta-
tion does not account for top-level container object classes (such as a window in
Windows95™, the rooms in [29] or the book in [30]). Instead, through sub-classing,
augmentation extends the range of simple or composite interaction elements that may
be supported in a toolkit’s object hierarchy.

In case that an alternative interaction metaphor needs to be supported, then it may be
realised through new toolkits. An example of the latter case is reported in [33,34] where
COMMONKIT is used to support user interaction based on a non-visual embodiment of
the Rooms metaphor through speech and/or Braille output and keyboard input.
COMMONKIT offers the full range of programming features encountered in currently
available GUI toolkits, such as hierarchical object composition, dynamic instantiation,
call-back registration and event-handling. In its current version, COMMONKIT imple-
ments only one container, namely Room, and several object classes (e.g. floor, ceiling,
front/back/left/right wall), in addition to conventional objects, such as menu, toggle
(represented as on/off switch), button, text reviewer, etc. A more elaborate account of
the object library of COMMONKIT, as well as applications built with it, can be found in
[34].
Following the above, the notion of a multiple metaphor environment implies a particular computer-based embodiment of an integrated system, capable of performing context-sensitive mapping between functions in a target domain (e.g. functions of a computer environment) to symbols in a source, or presentation domain (e.g. the desktop interactive embodiment) and vice-versa (Fig. 1). Alternatively, it may be conceived as an integrated multiple toolkit platform, capable of context-sensitive mapping. For example, consider typical functions such as file management, electronic mail and editing, as performed in a computer environment (target domain). Such functions in the target domain are mapped onto user operations on objects (i.e. folders, documents, drawers) of the source domain, namely the desktop.

The visual desktop embodiment in current computer systems, performs precisely such mappings between symbols from a target domain to symbols in the designated source domain. However, the visual desktop, as embedded in currently available user interface development environments, does not satisfy the conditions of multiple metaphor environment, since it does not perform any context-sensitive processing to map functions from the target domain to corresponding symbols in the source domain. This is because the source domain is fixed and unique (i.e. the desktop of an office). In other words, there is no possibility to map a file management function onto a book operation and vice versa. Consequently, the construction of multiple metaphor environments reflects two important properties, namely the explicit embodiment of alternative metaphors (i.e. desktop, book, library) into the user interface, as well as their fusion into an integrated environment (i.e. context-sensitive mapping).

To demonstrate the principles underpinning the design and development of multiple
metaphor environments, let us assume three users, namely a sighted user, a child and a blind user. All three are tasked to carry out a file management operation, namely delete a file. Since the capabilities of the users differ (e.g. with regards to the modalities that may be employed to facilitate the interactive task), the interface should ideally undertake the required transformation so as to present an appropriate (i.e. accessible and usable) instantiation, suitable for each user.

Fig. 2 depicts indicative examples of plausible alternatives, which can be realised in a programming-intensive manner, by providing separate interface implementations for each user. Alternatively, the same interface could be built, in such a way, so that it is capable of context-sensitive processing leading automatically to the undertaking of suitable transformations to map the file management operation onto appropriate interactive environments, such as those depicted in Fig. 2.

From the above, it follows that multiple metaphor environments are necessitated from the diversity of users (i.e. diverse requirements of different target user groups), the diversity of contexts of use (i.e. the variety of contexts in which artefacts may be encountered) and the diversity of interaction platforms (i.e. proliferation of different interaction toolkits), all of which may necessitate sometimes radical changes in the design. As a result, the important features characterising such environments are that: (a) there is a clear separation between knowledge and presentation; (b) the system integrates components (i.e. toolkits) implementing alternative interactive embodiments of a particular artefact; (c) the system is capable of performing context-sensitive processing and selection of suitable symbols to interact with the user, based on information provided by a dedicated tool usually referred to as user modelling component, or user information manager, offering information, both general and task specific, on the current user; (d) multi-modality is preserved through the fusion of metaphors into an integrated environment.

4. Building multiple metaphor environments: the unified interface development method

Achieving context-sensitive processing requires a mechanism for obtaining a global understanding of the task execution contexts. This, in turn, requires a broad view of how tasks are accomplished by different users across different presentation systems (or interaction platforms) and contexts of use. For that purpose, a method is needed to allow capturing alternative embodiments of artefacts depicting the diverse contexts of use that may be encountered (i.e. variation in users, platforms, environment). Additionally, suitable development support is required to allow such artefacts to be specified and realised. To address the above challenges in the context of designing and implementing user interfaces for diverse user groups, including people with disabilities, we have developed a methodology and a set of tools to facilitate unified user interface development [35].

A unified user interface is defined as a hierarchical construction in which the root represents an abstract design pattern de-coupled from the specifics of the target user (group) profile and the underlying interface development toolkit, while the leafs depict concrete physical instantiations of the abstract design pattern. The unified user interface development method comprises design- and implementation-oriented techniques for
accomplishing specific objectives. The design-oriented techniques (*unified user interface design*) aim towards the development of rationalised design spaces (see next section), while the implementation-oriented techniques (*unified user interface implementation*) provide a specifications-based framework towards constructing interactive components and generating the run-time environment for a unified interface.

4.1. Unified user interface design

The specific objectives of unified user interface design can be summarised as follows:

O₁: Accommodation of different user groups with varying abilities, skills, requirements and preferences, as opposed to the “average” able bodied user;

O₂: Elaboration and enumeration of different design alternatives into a design space and ranking of competing alternatives, based on agreed design criteria;

O₃: Articulation of abstract design patterns;

O₄: Propagation of design knowledge into the development cycle by embedding design recommendations into user interface implementation.

4.1.1. Designing for different user groups (user profiling)

To accomplish this objective, the design team recruits exploratory design techniques from human factors and user-centred interface design to identify abilities, skills, requirements and preferences of the target user groups. Thus, the design team will typically collect data on user interaction requirements, user preferences, and the way in which users (are likely to) accomplish tasks in the existing (or envisioned) task organisation. Techniques which may be used to achieve the intended objective include primarily formative design input techniques and may range from observation and questionnaire-based surveys to envisioning through scenario-based design [36], user modelling (perhaps, using a cognitive model [37] and performance measurement [38]). In cases where some of the required information cannot be collected, then rapid prototyping techniques may be used for requirements elicitation.

An important outcome of this stage is the identification of a comprehensive and exhaustive list of user parameters which shape design and determine the interaction of the target user groups with the interactive application for which the envisioned interface is being developed. A technique and a supporting tool environment for developing context-bound taxonomies of user parameters and user profiling in the area of disability is reported in [39]. Finally, in the context of developing multiple metaphor environments, this stage contributes to the population of the user modelling component, or user information manager that was mentioned earlier.

4.1.2. Enumeration of design alternatives

Users are not the only source of diversity that has an impact on user interface design. Interaction platforms and the context of use may also necessitate alternative interface designs. A pre-requisite for context-sensitive processing in an intelligent user interface is the enumeration of such plausible design alternatives. This will enable the interface to obtain the global understanding of the task execution contexts and to exercise the required
control upon detection of changes in status. Towards this end, the designer may employ techniques for analytical design (such as design scenarios, envisioning, ethnographic methods, etc.) to facilitate the identification of plausible design alternatives (e.g. the design space) for different user groups (see Fig. 3).

Design alternatives may be related to the semantics of interaction, the syntax of the dialogue and the physical interaction facilities used to realise the dialogue. At the semantic level, design options unfold from studying the expected, or desired attitude and characteristics of the overall interactive environment and deciding on the choice of suitable interaction metaphors (see for example the alternatives depicted in Fig. 2).

Syntactic issues may relate to the tasks the user has to perform, their sequence, as well as dialogue specific attributes, such as command order and function activation modes. Fig. 4 illustrates two examples depicting alternative command orders (e.g. object-function and function-object syntax) for deleting a file. In Fig. 4(a), the user first selects the object (e.g.
File1.txt) and then issues the command (e.g. Delete). The opposite syntax is used in the example of Fig. 4(b). Finally, at the physical level, the range of issues is equally broad and may range from selecting appropriate interaction objects to attributing specific interactive behaviour by articulating attributes of these interaction objects. As an illustration, consider Fig. 5 which enumerates several alternatives for deleting a file (in addition to those of Fig. 4).

From these examples, it becomes evident that enumerating interface alternatives creates large design spaces which, in turn, pose the challenge of devising suitable methods for managing them. To this end, unified user interface design employs abstraction and rationalisation to organise the design space and facilitate context-sensitive selection.

4.1.3. Propagating design recommendations to user interface implementations

In this phase (or design step), alternatives identified as a result of the previous phase are consolidated into abstract design patterns and, subsequently, the design space is rationalised. Design space rationalisation entails the provision of a principled mechanism for mapping an abstract design pattern into a concrete artefact. In what follows, we examine both the handling of abstraction and the means for rationalising the design space.

The technique used to identify and document abstract design patterns is called polymorphic task decomposition and it has been described in [40]. Here, we only provide a brief and informal account of the technique. According to the notion of polymorphic task decomposition, a task in a hierarchy of tasks and sub-tasks may be decomposed into an arbitrary number of sub-hierarchies depicting requirements, or preferences of different
Fig. 6. Decomposing abstract design patterns based on design criteria.
user groups. The leaf nodes of the hierarchy map to the enumerated design alternatives identified as part of the exploratory design stage.

An example of a simple polymorphic task decomposition and its relationship with design objectives and alternatives within a design space is depicted in the diagram of Fig. 6. The diagram refers back to our previous example of command order in direct manipulation user interfaces and provides a conceptual illustration of how an abstract design pattern may be mapped onto concrete alternatives based on designated criteria.

It should be noted that the concrete alternatives depicted in the right hand-side of the diagram exhibit both syntactic and lexical differentiation. In the unified user interface design method, such a blending of syntactic and physical aspects into a concrete design artefact is called a style [40,41]. For the same task, several styles may be concurrently available. Moreover, new styles (e.g. design updates) may also be introduced as requirements change, or as additional information becomes available. Design updates are a distinctive property of the unified user interface design method and a central notion in the polymorphic task decomposition. The technique used to facilitate design updates is called incremental design and may be informally conceived as a spiral process of progressively enhancing a design space by encountering and integrating new alternatives suitable for new contexts of use.

Incremental design features the notion of task context, which is used to collect and structure physical, syntactic and semantic knowledge pertaining to an interface element. A task context is a collection of knowledge-based descriptions of an abstract design component. The designer may populate several instances of a task context to reflect alternatives in the design space (e.g. styles). In the tools described in [39,42], incremental design is supported by providing an “empty” task context (referred to as Any_Other), which can be populated to reflect a new interface style. In practice, this means that any abstract design component has, at any time, at least one child, namely, Any_Other (see example in Fig. 6).

Navigation down to the leaf nodes of the polymorphic task hierarchy is guided by the association of design criteria, which substantiate the differentiated artefacts. In the example of Fig. 6, the criteria are derived from [43]. Specifically, the study by Kunkel et al. [43] reveals that the two alternatives for command ordering are indifferent across a number of independent variables (referred to as criteria in the context of our work), including task completion time, error time, efficiency of task performance and frequency of errors. Instead, with regards to action time, Function-Object syntax is preferred to Object-Function syntax, while with regards to planning time, the opposite holds true. This means that, the criteria for which a preference ranking can be specified are action time and planning time. In contrast, for the remaining criteria the two options are indifferent.

Eliciting such data, either from the available design wisdom, or by conducting suitable experiments, facilitates the development of preference profiles for different interface elements. Preference profiles in turn, allow the designer to rationalise the design space by generating indifference classes amongst alternative options, based on designated criteria. In terms of run-time behaviour, preference profiles provide a computational mechanism, which facilitates context-sensitive processing towards rational selection of maximally preferred options. A technique which implements such design space rationalisation based on preference and indifference relations has been described in [44]. It is important to mention that similar information may be collected for a broad range of
interaction elements pertaining to the semantic level (e.g. choice of interactive metaphor), syntactic properties of a dialogue (i.e. function activation mode, guidance), as well as physical attributes of interaction object classes (i.e. input/output devices, interaction techniques, feedback modalities, etc).

4.1.4. Propagating design recommendations to user interface implementations

One of the distinguishable objectives of the unified interface design method is that it seeks to provide mechanisms and constructs for embedding design knowledge into user interface implementations, thus making design decisions directly accountable during user interface implementation. The basic idea is that a relevant subset of the results of a design scenario are consolidated (either manually, or through computer-based support) into a collection of explicit assignments about elements of interaction. These elements may pertain to either the semantic level of interaction, the syntax of the dialogue (i.e. command order, function activation modes, etc), or the physical level of interaction (i.e. interaction object classes and their attributes).

There are two critical considerations regarding the content and scope of such assignments. First, regarding the content, it is crucial that these assignments should be interpretable and applicable by the run-time libraries of a user interface development toolkit. Secondly, the scope of these assignments should be bound to specific instances within the polymorphic task hierarchy (e.g. task contexts) and the corresponding interaction platform on which this instance of the polymorphic task hierarchy is to be realised. Both these requirements are currently met by compiling design recommendations per interaction object class attribute, object class, task instance and interaction platform. The general
format of these recommendations is defined by the predicate:

$$\rho(I, T, O, A, \text{assignment})$$

where I stands for the interaction platform (that provides the interactive embodiment of the metaphor), T is a task context in the polymorphic task hierarchy, O is the object class, A is the attribute of the interaction object class that is to be assigned as a result of the recommendation, and assignment is the value designated to this attribute by the recommendation.

To assist designers in populating a polymorphic task hierarchy with recommendations such as the above, a supporting tool environment has been developed, called USE-IT. The tool’s architectural properties, interactive environment and subjective usability evaluation have been described in [39], while the knowledge-based techniques and inference methods developed to facilitate the derivation of such recommendations are elaborated in [42].

4.2. The unified user interface development platform

USE-IT is a design-oriented component, which inter-operates with other software tools of the unified user interface development platform (see Fig. 7). The basis for such inter-operation is a collection of unified object classes and a suitable protocol for tool communication. The unified object classes are generated using a dedicated tool, called Platform Integration Module (PIM) [45], tasked with the integration of different interaction platforms. Integration in this context entails abstracting from the platform-specific properties and generating a common (unified) programming layer for utilising the physical interaction elements of the platforms. This unified programming layer enables interface developers to establish the desired programming constructs on top of the toolkits and eliminates the requirement to make direct calls to toolkit specific interaction elements when developing the interface specification.

Once the recommendations have been compiled (by USE-IT) and stored in the recommendations server (see upper part of Fig. 7), then the user interface specification can be developed using a specification language, called G-DISPEC [46]. G-DISPEC enables the interface developer to generate a user interface implementation from specifications rather than by programming. Such a specification, through dedicated functions of the Application Programming Interface (API), can interrogate the server (see step 1 and step 2 in Fig. 7) to collect all recommendations relevant to a particular object class (i.e. menu) for a particular task context (i.e. convey warning) and interaction platform. The application of these recommendations is realised by tools, which link with the target platform as opposed to making direct calls to the platform [45,46].

5. Experience with the unified user interface development method

The unified user interface development method has been applied in some application domains in the context of collaborative research and development work. Initially, it was validated in two case studies, namely, the development of a hypermedia application for blind users [47] and the development of two interpersonal communication aids for speech-motor and language cognitive impaired users [48]. Both these developments were
undertaken in the context of the ACCESS project (see Acknowledgements) and entailed
the use of the tools of the unified user interface development platform by ACCESS
consortium partners to construct the user interface components of the applications.

Subsequently, the unified user interface development method was applied to design and
implement accessible browsers for Web-based interaction with a metropolitan information
system. The system, developed in the context of the ACTS-AVANTI project (see
Acknowledgements) is targeted for the population at large, including people with disabil-
ities [41]. Through these projects and the acquired experience, several lessons have been
learnt with regards to the relative merits and implications of the approach.

5.1. Design implications

First of all, the unified user interface design method adopts a middle-out [49], as
opposed to bottom-up, or a top-down approach to design. This is deemed as a relative
merit of the technique, as designers rarely work in a top-down, or bottom-up fashion.
Another advantage of unified user interface design is that it runs orthogonal to traditional
design approaches, such as human factors evaluation, cognitive science and user-centred
design, as far as rationalisation of design spaces is concerned. In other words, the designer
may select any technique considered suitable to test alternative styles and experiment with
design alternatives. The only requirement of the unified user interface design method is
that such results should be subsequently interpreted into the representations used by the
tools of the unified user interface development platform. This ensures the seamless propa-
gation of design specifications into user interface development cycles, thus bringing
disciplined design closer to user interface implementation. Moreover, the inter-operation
of the design support environment with the user interface development tools reduces the
problem of guaranteeing compliance of the implemented artefact to specifications, or
misunderstandings, which frequently occur when design concepts are communicated to
interface developers.

5.2. Implications on the implementation of user interfaces

Unified user interface development raises several requirements regarding the under-
lying development environment. Specifically, a unified user interface entails a single
implementation, targeted to the broadest possible end user population. The main properties
of this unified implementation are two-fold. First of all, it does not involve any direct
“calls” to the target platform(-s) or toolkit(-s). Instead, it utilises specific functionality, or
tools (e.g. technology/platform servers) to connect with the underlying platform(-s) in an
independent manner. Secondly, during interaction, abstract dialogues are mapped to
physical dialogues by making use of recommendations compiled by orthogonal design
tools. Physical dialogues may represent alternative patterns of behaviour, either at lexical,
syntactic or semantic layers, depending on the requirements of the target user (group).

In order to provide for the above, a user interface development environment should
make provisions for three important requirements, namely, toolkit integration, toolkit
augmentation and toolkit abstraction. These requirements may be attained either through
dedicated tools, such as those comprising the unified user interface development platform
depicted in Fig. 7, or alternatively, through a programming-intensive approach which, however, is bound to be less effective.

5.2.1. Toolkit integration

It is frequently encountered that the interaction facilities of a particular toolkit do not suffice to provide implemented versions of the desired range of design artefacts. For instance, the animation facilities embedded in a particular toolkit may not be of the type required by an application and, therefore, there may be a need to integrate and use an add-on animation library. In a similar manner, in order to implement a unified user interface which is concurrently accessible by different user groups (i.e. sighted and blind user), it may be required that the visual and non-visual faces of the interface are respectively mapped onto visual and non-visual toolkits. From these examples, it becomes evident that the requirement to use interaction facilities from different toolkits is important in order to address diverse users and contexts of use. Toolkit integration captures precisely this requirement and entails that a development platform offers the facilities required to hook new interaction elements from different toolkits. Referring back to our example of sighted and blind users, toolkit integration requires that, for example, the Windows95™ object library and the COMMONKIT non-visual toolkit [26,33,34], are integrated so as to facilitate alternative embodiments of interactive artefacts, suitable for the sighted and blind users respectively.

5.2.2. Toolkit augmentation

In order to facilitate access to a windowing system, such as Windows95™, for a motor impaired user, it may be required that the basic object classes of a particular toolkit are enhanced, so as to enable additional interaction techniques to be embedded into the foundation object class library. Thus, for example, an augmentation of the basic Windows95™ object libraries may be realised, through sub-classing, which allows scanning to become an integral property of the overall interactive environment. Scanning refers to a technique for selecting options (i.e. characters, words, symbols etc.,) from a target selection set using one, or more switches. Scanning can be manual (i.e. movement of cursor or highlighter is initiated by the user through the switch) or automatic (i.e. cursor or highlighter automatically moves from one option to another within the target selection set,
in predetermined time intervals). The resulting interactive behaviour allows a disabled user to accomplish tasks (such as for example text editing), using dedicated interface components.

In the example shown in Fig. 8, two instances of augmentation of Windows95™ are depicted. On the left hand side, the toolbars illustrate an enhancement of window management facilities, so as to allow a user to carry out window management through scanning. The second example on the right hand side, depicts a virtual keyboard which automatically pops-up when the focus is on a text entry field. This allows a user to edit text fields using scanning. It is worth pointing out, that there are various virtual keyboard layouts from which the user may select the one that best suits specific requirements. More details on these examples and the dialogue that implements the augmented environment can be found in [32], for the case of Windows95™, and in [41] for Web-based interaction. It is important to mention that in both cases, the augmented interaction facilities become integral part of the overall environment (e.g. Windows95™). Thus, developers may reuse them across different applications without any additional development overhead. This should be assessed against current practice, in which scanning can only be integrated through programming, thus increasing the cost of development and maintenance of the user interface, while also limiting reusability across applications.

5.2.3. Toolkit abstraction

This entails that the developer is provided with a unifying toolkit programming layer which hides the specifics of each toolkit and minimises the associated programming burden. Thus, for instance, alternative embodiments of a selection task (see Fig. 9) that
may be supported by different interaction platforms (or toolkits) can be consolidated into a *unified selector class*. Such an abstract object class would only require two attributes, namely the *number of options* and *user selection*, thus being totally relieved from the physical properties of any particular toolkit. Moreover, at run-time, it may be mapped onto a suitable concrete (toolkit-specific) object class, provided that the server of the designated toolkit has been integrated. Towards this end, a tool called Platform Integration Module (PIM), was developed to provide facilities for establishing alternative programming layers on top of existing toolkits [45].

The PIM tool provides a special-purpose language for the specification of the desired programming structure of interaction elements for the new programming layer. Such a specified structure may follow the minimal interaction object modelling, according to which the properties of objects are logically split in three main categories, namely input techniques and attributes per input technique, output techniques and attributes per output technique, and behaviours and attributes per behaviour. Additionally, container objects have three more attribute categories, namely, topology policy, access policy and navigation policy. These attributes exemplify how the objects within the container are organised (e.g. horizontally, vertically, by name), how they are accessed once the focus is on the container (e.g. button press, mouse selection), as well as how navigation within the container is to be accomplished (e.g. arrow keys, next/previous buttons).

Toolkit abstraction is a critical component of any multiple toolkit platform. When appropriately supported, it substantially eases the development effort and provides a unifying view of interaction elements across different toolkit platforms. As one might expect, the utility curve associated with toolkit abstraction is similar to that of groupware [50]; as the number of integrated toolkits increases so does the added value of toolkit abstraction capabilities. Though recent literature offers few examples where the toolkit abstraction requirement has been fully supported within a development environment, there are several examples of developments in this direction. For instance the notions of interactors, meta-widgets and virtual objects depict cases of the trend towards toolkit abstraction. It is expected that, as the need for multiple toolkit platforms increases to accommodate the development of radically different implemented artefacts, more efforts will progressively be devoted to the study of toolkit abstraction.

These requirements have been the primary driving forces for developing the unified user interface development platform (of Fig. 7) to couple the unified interface design approach. A more elaborate description of this platform is given in [35], where specific aspects of the development process are presented.

### 6. Discussion

The unified user interface development method was not developed through an explicit effort to advance Intelligent Interface Technology, but as an attempt towards the construction of multiple metaphor environments, accessible and usable by the broadest possible end user population, including people with disabilities. However, it is argued that the method provides an alternative insight to the design and development of intelligent user interfaces and an example of the reality of this technology in the context of the emerging
communication-intensive paradigm for computer-mediated human activities. Specifically, unified user interface development provides a novel perspective on the architectural abstractions for intelligent user interfaces which explicitly accounts for diverse user groups with radically different interaction requirements.

In the past, there have been several attempts to extract a reference model from concrete user interface architectures in order to classify existing prototypes and to guide the construction of user interface software. The best known architectural abstractions of user interface software include the Seeheim, PAC, ALV, MVC and Arch/Slinky models [51–53]. However, they do not consider user interface adaptation, or intelligent interaction aspects. Moreover, the literature on intelligent user interfaces, offers specific examples as opposed to general architectural abstractions for intelligent systems [5]. Even those, however, limit their scope to address adaptivity, which constitutes only one dimension of user interface adaptation in interactive software systems. They do not explicitly account for the equally important objectives of accessibility and high quality interaction, by users with diverse requirements and, as a result, do not provide design support to facilitate the study of context variety which is the driving requirement for embedded intelligence.

Through experience with the unified user interface development method and by constructing user interface software tools to support the design and development phases, we have been exposed to a wide range of issues regarding the architecture of intelligent user interfaces. Although, a detailed analysis of these issues is beyond the scope of this article, we will be content to provide a summarising account of some key requirements. First, it is argued that intelligent interactive behaviour should be driven from the integration of orthogonal user interface software components, inter-operating according to well defined semantic and message passing protocols. Orthogonality entails, amongst other things, that a user interface development system can utilise (the results or services of) external tools in order to determine, at run-time, the type of interactive behaviour that is to be instantiated and thus, accomplish context sensitive processing.

Secondly, it is important that interactive behaviour is generated from specifications as opposed to programming. Reducing implementation to specification introduces several challenges including the development of powerful specification techniques capable of handling, amongst other things, syntactic and lexical polymorphism. With regards to the phase of generating the user interface implementation, it needs to become more sensitive to design, allowing design principles and recommendations (such as those compiled by USE-IT) to be directly accounted for and embedded into user interface implementation.

Accessibility by different user groups and high quality interaction raise additional issues with regards to how an intelligent interface is to be developed. The novel contribution of the unified user interface design to the study of accessible and high quality interactive software is that accessibility and interaction quality are directly addressed by specific tools and techniques (e.g. polymorphic task hierarchy). To this effect, unified user interface design advocates the encapsulation of alternative accessible interactive behaviours into abstract and reusable design patterns. Additionally, the unified user interface development platform provides the required support for realising accessible and high quality interactive software. This is achieved by providing a suitable run-time environment for the instantiation of the most appropriate design alternative within a design space, so as to facilitate the
requirements of a particular user and a given context of use. This functionality is not only additional (and complementary) to the convention of dynamically enhancing the interaction with a single implemented artefact, but most importantly, it extends the scope and range of interactive behaviours that can be supported.

7. Summary and conclusions

This article has described an approach which emphasises a particular design and development perspective upon the construction of intelligent interactive software, resulting in user interfaces that exhibit, in addition to adaptive behaviour, accessibility and high quality interaction. The underlying framework which was used to develop the article’s argumentation is based on the unified user interface development method. This method comprises a collection of design and implementation techniques, which facilitate the construction of user interfaces accessible and highly usable by the broadest possible end user population, including people with disabilities.

Specifically, unified user interface design offers an analytical framework for enumerating plausible design alternatives, articulating reusable design abstractions and rationalising a design space on the grounds of suitable criteria and design rationale. In contrast, the unified user interface development platform provides a sophisticated environment for implementing unified user interfaces. This environment makes use of several tools which allow: (i) the construction of a unified user interface as a composition of abstractions at different levels; (ii) the manipulation and management of the physical (toolkit-specific) resources; and (iii) the establishment of the relationships between the involved abstractions and the available physical resources.

The article reflected upon recent practice and experience with the unified user interface development method and indicated how this method links with existing design approaches, as well as how it may be employed to facilitate the development of intelligent interactive systems, which account for different user groups, including people with disabilities. In particular, it was described how a unified user interface artefact can be encapsulated, articulated and populated in terms of design recommendations. We also briefly reviewed the key requirements for tools allowing unified user interface implementation and the way in which design recommendations are embedded into user interface implementations.

The main conclusions from the present work is that designing and implementing user interfaces for radically different user groups requires a shift in prevalent HCI practice. Such a shift should be characterised by the development of: (i) analytical design approaches capable of capturing how tasks are accomplished by different user groups; and (ii) specification-based frameworks for user interface implementation. Analytical HCI design should spread the focus of design activities both on the artefacts being constructed as well as the reasoning behind them. However, specification-based user interface development environments should progressively replace the currently prevalent programming-intensive approach towards the development of interactive components.

With regard to Intelligent Interface Technology, the current work offers several contributions. First of all, the notion of a multiple metaphor environment provides a new
conceptualisation of how intelligent and co-operative interactive behaviour can be attained, through context-sensitive processing, and integrated into an interactive system. Secondly, the unified user interface design method offers an insight into how diversity can be captured and consolidated in rationalised design spaces, which facilitate context-sensitive processing. Finally, the unified user interface development platform contributes towards an improved architectural abstraction for intelligent user interfaces and details how such abstractions can be realised in high level user interface development environments.

On-going and future work seeks to further advance the unified user interface development approach to take account of recent developments in enterprise computing (e.g. distributed and component-based computing) and novel user interface software technologies (e.g. augmented and virtual realities, wearable equipment). This will shed additional light into the economic efficiency and efficacy of the proposed approach in the long run.

Acknowledgements

The development of the unified user interface development method has been supported by collaborative research and development projects partially funded by the European Commission (DG XIII). These projects are:

The TIDE-ACCESS (TP 1001) project (Development Platform for Unified Access to Enabling Environments). The partners of the ACCESS consortium are: CNR-IROE (Italy)-Prime contractor; ICS-FORTH (Greece); University of Hertfordshire (United Kingdom); University of Athens (Greece); NAWH (Finland); VTT (Finland); Hereward College (United Kingdom); RNIB (United Kingdom); Seleco (Italy); MA Systems & Control (United Kingdom); PIKOMED (Finland).

The ACTS-AVANTI AC042 project (Adaptable and Adaptive Interaction in Multi-media Telecommunications Applications). The partners of the AVANTI consortium are: ALCATEL Italia, Siette division (Italy)-Prime Contractor; IROE-CNR (Italy); ICS-FORTH (Greece); GMD (Germany), VTT (Finland); University of Sienna (Italy), TECO Systems (Italy); STUDIO ADR (Italy); MA Systems and Control (UK).

References


