Multiple Metaphor Environments: designing for diversity

D. AKOUMIANAKIS† and C. STEPHANIDIS†‡*

†Institute of Computer Science, Foundation for Research and Technology-Hellas, Science and Technology Park of Crete, Heraklion, Crete GR-71110, Greece
‡Department of Computer Science, University of Crete, Crete, Greece

Keywords: Context-sensitive processing; Source domain; Target domain; Interaction.

This paper advances a proposition for the engineering of interactive computer-based environments capable of exhibiting alternative interactive embodiments to cope with diversity in users, interaction platforms and usage contexts. Such systems are referred to as Multiple Metaphor Environments (MME). The theoretical underpinnings of an MME rely on a conception of HCI design as mapping functions in a machine-oriented language (target domain) to symbols in a user-oriented language (source domain), and vice versa. Such a conception, which is rooted in developments in communication theory and the philosophy of language, constitutes the baseline for formulating a proposal for the design of MME. The proposal comprises a set of engineering principles, process-oriented guidelines and design techniques intended to facilitate a detailed account of how interactive systems could be designed to cope with diversity. To aid the articulation of the various properties of MME, we refer to concrete case studies that provide exemplars of novel insights and promising design practices towards the specification of MME.

1. Introduction

Human–Computer Interaction (HCI) researchers have long been concerned with the study of metaphor and its use as a prime instrument towards improved usability. In general, metaphors are offered to illuminate their referents’ meaning; they are most useful and most successful in this respect when they associate an unfamiliar and/or abstract referent to a familiar and/or concrete concept. In HCI, the rationale for designing with metaphors lies in the notion that the overall cognitive fit between a user and a system may be substantially improved if the user interface presents an embodiment close to the user’s experiences, prior knowledge, competence and expectations. By implication, learning, performance and user satisfaction can be improved. In the past, several HCI innovations were conceived and popularized as a result of effectively articulating a suitable metaphor and embedding it into software design. Examples include the Xerox Star developed at Xerox in 1981, which introduced the visual desktop embodiment of the computer, the electronic spreadsheet invented by M. Kapor, the founder of Lotus, and many others.
Today, as the number and diversity of computer-mediated activities exceeds the boundaries of the conventional desktop, the need for systematic articulation of novel metaphors becomes compelling. The reasoning behind this is rather straightforward. As users interact with a computer system, they make use of prior schematic knowledge to develop a mental model of the system. HCI designers should therefore seek to provide designs that offer conceptual models of the system that fit cognitively with the user's mental model. A common means for aiding the formation of accurate mental models is to incorporate a metaphor into the user interface design (Carroll and Mack 1985). This is justified by the conception of metaphors as common vocabularies (of terms, images and concepts) that users are already familiar with. Therefore, metaphors can assist users in developing accurate mental models of a system by allowing the organization and association of new information through already acquired knowledge. In other words, metaphors provide for new information to be interpreted and used within a previously existing knowledge structure of users.

Nevertheless, the choice of suitable metaphors to characterize modern applications becomes more complex due to intrinsic properties of the prevailing computational paradigm. Specifically, computer networking, the World Wide Web and the distribution of knowledge across geographic locations enables a new range of computer-mediated human activities and changes the content of tasks. The desktop environment is no longer the one common denominator of computer users, while the range and type of devices available exhibit radically different properties (e.g. input/output devices, interaction techniques, information-processing capabilities). At the same time, digital content is growing rapidly resulting in a wide range of computer-mediated human activities, thus increasing the potential users and uses of such content. It stands therefore to reason that the gains to be experienced from constructing novel, intuitive virtualities to characterize the emerging Information Society are tremendous. (The term virtuality refers to any kind of technological construction that allows humans to attain business, residential or communication-oriented activities. For example, in the business environment, the desktop has been the prominent virtuality. However, in the future, and given the collaboration- and communication-oriented paradigm that progressively emerges, new virtualities, e.g. digital cash, Internet communities, etc., are likely to prevail and shape the conduct of computer-mediated human activities.) At the same time, as pointed out in Stephanidis et al. (1998), the challenge to designing such virtualities is unprecedented owing to plethora of interaction platforms, novel contexts of use and the radically different requirements of an increasingly wider client base.

In this paper, we seek to advance an understanding on the design of computational systems that can provide intuitive and natural interactive embodiments of computer-mediated human activities. Our normative perspective is based on the premise that no single metaphoric design option is likely to be good for all tasks and all users in all contexts of use. (This is also evidenced from the fact that no research effort aiming to explore a metaphor in the context of HCI makes any claims about optimality of the choice of metaphoric design.) Thus, for example, a library system should present an alternative, and perhaps radically different, image to a young computer-illiterate pupil from that presented to an adult expert user. Consequently, future interactive computer-based applications should increasingly fuse metaphors to accommodate diverse user requirements and radically different contexts of use. The term used to refer to such systems is Multiple Metaphor Environments (MME), which implies a capability on behalf of the system to adapt to
a changing environment. Specifically, MME constitute our vision of a new type of integrated system capable of performing context-sensitive processing for mapping functions in a machine-language to symbols in a suitable user (or presentation) language and vice versa. The term MME originates from earlier research efforts in the Japanese FRIEND21 Project (Institute for Personalized Information Environment 1995). In Akoumianakis et al. (2000), we reformulated the original concept in the context of Intelligent Interface Technology. Our intention here is to extend previous conceptions of MME and provide engineering ground and principles for the design of such systems. To this end, the paper seeks to establish a set of key properties and extrapolate towards the translation of these properties into guidelines and recommendations for constructing systems exhibiting the properties of MME.

The paper is structured as follows. The next section defines MME and unfolds the rationale for such systems in the context of the emerging information society. Then, the paper presents guidelines and techniques for the design of such systems and provides brief accounts of relevant design case studies. The discussion section distils the characteristic elements of an MME and provides an informal account of their implications on prevalent HCI conceptions. In particular, we draw attention on the need to capture the global execution contexts of the tasks a system is designed to facilitate, as well as the need for a specification- as opposed to programming-oriented frame of reference for realizing MME. Finally, the paper ends with a summary of key contributions and conclusions.

2. MMEs: definition and rationale

The notion of an MME 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. Such mapping is illustrated in figure 1, which shows three main logical components of MME: source domain functions, target domain symbols and context-sensitive processing. In this section, we briefly review each component, while more elaborate accounts are provided in subsequent parts of the paper.

2.1. Target domain functions

The term ‘target domain functions’ refers to typical computational operations such as file management, electronic mail, data storage, database transactions, etc. as performed in a computer environment (target domain), which may be a personal desktop computer, a mobile device or any other information appliance. Typically, such functions are realized through complex device-specific operations and processing, which may differ from one device to another. Moreover, different devices need not support the same type or range of functions. Consider, for instance, the traditional PC, a WAP phone and a Personal Digital Assistant (PDA). From their conception, these devices were designed and constructed to facilitate different type of human activities through dedicated functions.

2.2. Source domain symbols

In current systems, such functions in a target domain are mapped onto user operations on objects (i.e. folders, documents, drawers) of a source domain, e.g. the graphical desktop. In the past, HCI research has explored various source domains, which are briefly explored below, with the graphical desktop predominating in both
business and residential environments. With the advent of the Internet and the availability of mobile devices such as WAP phones and PDAs, the suitability of the graphical desktop to continue to penetrate developments became questionable. Nevertheless, in the case of some devices, such as PDAs, which have their origin in information-centric devices such as desktop computers, the Graphical User Interface model remains the prime source domain.

It is important to note that the graphical desktop embodiment in current computer systems implements and performs the mapping between functions in a target domain to symbols of a designated source domain. Nevertheless, irrespective of its embodiment, the graphical desktop does not satisfy the conditions of MME 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. In other words, there is no possibility whereby following some sort of data processing, to map a file management function onto a book operation and vice versa. Consequently, context-sensitive processing is a critical component of an MME.

2.3. Context-sensitive processing

Context-sensitive processing in an MME serves the purpose of implementing the mapping between functions in a target domain to symbols in a suitable source domain and vice versa. It should be noted that such context-sensitive processing can facilitate all three categories of context-aware functions defined by Dey et al. (2001, pp. 108–109), namely presenting information and services, automatically executing a service and attaching context information for later retrieval. In addition, it accommodates a strong commitment towards interoperability to convey the view
that functions of a target domain (i.e. data storage) can be mapped to symbols of alternative source domains and vice versa. The exact realization of such mappings (i.e. which function is mapped to what symbol) is determined through processing of a range of parameters, some of which fall under the category of ‘context information’, according to the definition of Dey et al.

From the above it follows that the construction of MME should reflect two important properties: the explicit embodiment of alternative interactive task manifestations or metaphoric realizations (e.g. desktop, book, and library) into the user interface, as well as their fusion into an integrated environment. Such an integrated environment could subsequently identify the need for and undertake context-sensitive processing to implement the mapping of target domain function(s) to the appropriate source domain symbol(s) and vice versa.

3. Properties of MME

Three important properties characterize MME.

- There is a clear separation between content and presentation.
- The system integrates components (i.e. toolkits) implementing alternative interactive embodiments of a particular artefact (e.g. support for multiple design languages).
- The system is capable of performing adaptations (both at the level of content and the level of presentation) through context-sensitive processing and selection of suitable symbols to interact with the user; these adaptations should be based on information provided by dedicated tool(s) offering information, both general and task specific, on the current user.

The following sections briefly elaborate on the above three properties with the intention of highlighting specific design requirements, their rationale and the techniques through which the requirements can be met.

3.1. Separating content from presentation: target and source domains

The premise of MME is the capability to perform context-sensitive mappings between functions in a machine-oriented language to symbols in a user-oriented language and vice versa. Such a premise calls for a clear separation between functionality (as implemented by non-interactive software components in a machine) from presentation (as implemented by the user interface). Though such a claim is not new in the HCI literature (e.g. Pfaff 1985, The UIMS Tool Developers Workshop 1992), it is of paramount importance for the conception and implementation of MME. As a result, we will briefly describe the need for this separation and the meaning it obtains in the context of the present work.

Human information processing relies heavily on interpreting and making inferences about the environment on the basis of existing beliefs, purposes and intentional structure with respect to objects in the environment. Thus, for an object to have a meaning, it must be interpreted against background information, which is usually referred to as context. According to Dey et al. (2001), context is ‘any information that can be used to characterize the situation of entities (i.e. a person, place or object) that are considered relevant to the interaction between a user and the application’ (p. 106). Winograd (2001) makes an even stronger argument that ‘something is context because of the way it is used in interpretation …’ (p. 405).
Despite of how broad or specific the definition of context might be, it is commonly agreed that the context-sensitive interpretation capability of humans has not been an intuitively observable characteristic of symbolic information processing by machines, as the latter usually does not take context explicitly into consideration (Day et al. 2001).

For humans, the computer is yet another object in the environment. This means that in order for the computer to have a meaning, it must be interpreted in a context. However, since humans and machines differ with regards to the way in which information is processed, it is necessary to provide ‘bridges’ across the two domains. For this purpose, the present work distinguishes between tasks, which correspond to the functional core of an application, and images corresponding to the interactive components of the application, respectively (figure 2).

Tasks refer to work items performed (or to be performed) by humans. They are machine-oriented (e.g. they are represented in a machine-oriented language which facilitates processing). As a result, they are function- or content-specific and constitute the target domain. Humans, due to different information processing patterns, may not sense or perceive these tasks, unless they are conveyed as images (i.e. concrete forms of symbol) representing the tasks or the state of the system, in a suitable user-oriented language (i.e. source domain). To this effect, images may take a number of forms, including icons, pictures, figures, text, music, voice, etc. It follows, therefore, that a particular task may be represented through alternative images depending on a designated choice of modality (Miller and Stanney 1997). In this context, the notion of a metaphor (Lakoff 1993, Ortony 1993) is relevant to the extent that it offers a structure mediating the representation of tasks by images. Thus, an image can be conceived as a metaphoric representation (figure 3).

Figure 2. Tasks and images.
As humans differ with regards to how they perceive objects in the environment, it is pertinent that the computer (and in particular the interactive components) should be able to convey tasks in alternative images (i.e. source domains) suitable for the target user group. At the same time, due to technological proliferation, a user task may be executed in alternative machine-oriented languages (i.e. target domains such as the personal computer, the television, the cellular phone, etc.), thus necessitating the capability on behalf of the system to map symbols in the source domain to functions in different target domains. To facilitate such mappings, the content (i.e. tasks) and presentation (i.e. images) should be separate.

In the context of HCI, the issue of presentation is of particular interest. This leads to the formulation of the second property of MME, which specifies how alternative presentation systems can be developed and articulated in engineering practice. To this effect, the notion of a design language is prominent, and constitutes the core of a proposed method for integrating alternative metaphoric representations as integrated components of a system.

3.2. Source domain management
3.2.1. Role of design languages: Whenever people construct objects, they draw upon a background of shared design knowledge in their community and culture (Winograd 1995). In HCI, design languages are widely used for interface design. For
example, all current GUI interfaces draw upon a basic vocabulary and interaction style that involves a set of standard design elements and tools (e.g. dialogues, menus, toolbars, window management) and supporting guidelines. Consequently, a design language can be defined as a mechanism mediating the mapping between functions of a target domain to symbols in a source domain and vice versa. Two properties of a design language are critical to consider. The first is the conceptual or ontological domain of the language, while the second is the computational manifestation of elements within this ontological domain through suitable interaction elements. The ontological domain of a design language corresponds to the underpinning conceptual metaphor (i.e. architecture) and provides the base ground of the language. In this paper, we will be concerned with the interactive embodiment of a design language into a computational environment. Usually, the interactive manifestation of a design language covers a limited subset of its ontological schemata. Moreover, the properties of the interactive components may not map one-to-one the properties of their physical (ontological) counterparts (Carroll et al. 1988). Thus, for example, whereas post-it notes are very typical elements of a physical desktop, they are typically not associated with a corresponding interaction object on the computer desktop.

We can therefore conclude that user interface development toolkits inherit the properties and implement functions of a particular design language, thus enabling developers to use the basic design vocabulary across different applications. Today, the only well-documented and widely adopted design language embedded into user interface development tools is that of the desktop. More recently, novel technologies, such as the WWW and virtual and augmented realities, have extended the basic desktop design vocabulary to accommodate elements from alternative design languages (e.g. links and browsing in WWW).

However, it is more than likely that in the future neither the desktop nor the prevailing WWW design language will suffice to provide appropriate computer embodiments for the broad range of emerging computer-mediated human activities. As Mountford (1990, p. 17) points out, ‘future interfaces need to incorporate new information types and to accommodate new types of users with additional customized real-world interface metaphors that make information easy to find and use’. Thus, it is important that designers can use alternative design languages to cope with diversity and the varieties of context. This, however, necessitates a coherent body of knowledge, which would guide designers to select appropriate real-word metaphors and systematically incorporate them into computer-embodied artefacts. Currently, such a body of knowledge is lacking, though there have been contributions towards the methodological study of metaphors in HCI (i.e. Carroll et al. 1988), exploration of alternative metaphors such the book (Moll-Carrillo et al. 1995, Card et al. 1996), as well as comparative studies of metaphors (i.e. Miller and Stanney 1997). To this effect, it is necessary to examine possible alternative strategies for articulating design languages into a user interface development toolkit, and the implications that this may have on existing or new applications built using such a toolkit.

3.2.2. Design language articulation: Following the prevalent conception of metaphor as characterizing either an element of the communication medium or the entire interactive episode (Ortony 1993), we introduce two prime strategies for articulating design languages into user interfaces. The first entails the use of a design
language to extract objects, which will characterize top-level containers in the user interface. The second strategy is to employ components of the design language to assign a metaphoric representation to non-top level container objects of the user interface. For example, let us consider the hierarchical object model, which is common in Graphical User Interface toolkits. A design language can be used to expand the root elements of the hierarchy (horizontal expansion), or alternatively to increase the depth of the hierarchy by introducing new lower-level elements (vertical expansion).

3.2.2.1. Horizontal expansion: This is the more challenging of the two and refers to the case where a totally new design language is deemed appropriate as an interactive embodiment of new functions. In other words, horizontal expansion entails the use of a design language to characterize the overall interactive embodiment of applications. In this case, the design language is used to determine the choice and interactive behaviour of the top-level container object classes of the user interface. Top-level containers provide the ‘hosts’ for interaction components, either primitive (e.g. interaction object classes) or composite (e.g. dialogues) in an interactive environment (Savidis and Stephanidis 1998). They usually appear as ‘roots’ in the hierarchical object model supported by currently prevalent user interface development tools.

Container object classes (see figure 4 for alternative interactive embodiments of a book container) require additional attributes to capture their interactive behaviour, than those typically associated with non-container interaction object classes. Typically, these attributes exemplify the navigation, access and topology policies associated with the container (ACCESS Consortium 1995). The navigation policy details how navigation within the container is supported. In other words, how the user may move from one interaction element to another within the container. Access policy details how interaction with the container is initialized as well as how distinct interaction elements within the container are accessed. Finally, topology policy relates to how the interaction elements are organized (e.g. horizontally or vertically).

For instance, the visual embodiment of the desktop, as experienced in Windows95™ and other graphical user interface development toolkits, presents the user with an interaction environment based on top-level containers, such as sheets of paper called windows, folders, etc., which characterize the overall interactive embodiment of the computer. Systems such as those shown in figure 4 and described in Moll-Carrillo et al. (1995) and Card et al. (1996), respectively, are

Figure 4. Alternate manifestations of the book.
examples of alternative embodiments characterized by different top-level containers, such as the book. Moreover, there have been other efforts, in which the primary top-level container was the room, e.g. Henderson and Robertson (1986).

In any case, the use of a design language to extract and implement suitable top-level container object classes is a non-trivial activity and several studies have proposed methodological insights to account for the complexity of the task (e.g. Carroll et al. 1988). For the purposes of this paper, it suffices to point out that developers need to consider a broad range of issues. These range from physical or surface-level aspects of interaction (e.g. layout and graphics), to interactive behaviours including navigation, access and topology policies as well as choice of interaction elements and components resident within the container.

3.2.2.2. Vertical expansion: This is the second strategy for articulating design languages. It refers to the case where new interaction elements (from the ontological schema of the design language) need to be introduced to a pre-existing collection of interaction elements (e.g. toolkit) which implement part of the designated design language. Thus, vertical expansion refers to the use of a design language to characterize objects embedded in the user interface (i.e. non-container object classes). This is shown in figure 5, which compares the horizontal and vertical expansion strategies. The dashed elements represent newly introduced interaction facilities. As shown, vertical expansion alters the object hierarchy at lower levels, compared with horizontal expansion, which introduces new top-level containers.

An example of vertical expansion is the menu interaction object class, as commonly encountered in popular user interface development toolkits. Making a choice from a menu is an activity that is usually encountered in a restaurant.

Figure 5. Contrasting horizontal and vertical expansion strategies for articulating design languages into user interface software.
Similarly, the computer-equivalent of the restaurant’s menu inherits such a role and provides a virtuality whereby the user is presented with a selection set from which he/she makes a choice. Nevertheless, what is important to mention is that such a virtuality does not characterize (or alter) the overall interactive embodiment of the user interface. This is evidenced from the fact that the menu as an interaction element is available in user interface development toolkits implementing radically different design languages.

3.3. Adaptation through context-sensitive processing

The final property of an MME is the capability to undertake adaptations through context-sensitive processing. Adaptations may be related either to the content or to the presentation, and may be initiated and realized by designers and developers, by the end user of the interactive system or by the interactive system itself. In all cases, a conscious design effort is required to identify, specify and evaluate plausible adaptations in relation to the intended context of use. MME facilitate adaptable and adaptive behaviour through context-sensitive processing.

Context-sensitive processing implies the capability on behalf of the system to reason towards the selection of the maximally preferred option(s), amongst a range of plausible alternatives, given the constraints characterizing the context of use. Compared with the conventional notion of user interface adaptation, as depicted in the HCI literature (Dieterich et al. 1993, Gong and Salvendy 1995), adaptation in MME has a wider meaning and scope. To illustrate this, we examine two scenarios that depict radically different cases and raise important implications for the design of systems complying with the contextual definition of MME. The first scenario builds on the notion of adaptability and depicts the case where the MME, through context-sensitive processing, selects the most appropriate interactive embodiment for a task. This choice is a selection amongst competing alternatives. The second scenario, which is more complex, describes a situation where the MME implements context-sensitive processing to decide on the fusion of interactive components from radically different design languages.

3.3.1. Adaptability: making a choice amongst competing alternatives: Consider the example of a visual desktop menu or a toolbar intended to convey choice amongst related items. A conventional user interface would present the menu on the screen and could also 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). Now 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. The user 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. In this case, context-sensitive processing would require the following.

- Identification of plausible alternatives to convey selection, such as the desktop menu and the 3D acoustic sphere for the blind (Savidis and
Stephanidis 1995), in anticipation of the variety in the context of use that the artefact is to be encountered.

- Unification of alternative concrete design artefacts into abstract design patterns or unified design artefacts (such a generalized container).
- Method to allow the assessment of alternatives in relation to the current objectives and the mapping of the abstract design pattern into the appropriate concrete/physical artefact.

A user interface complying to the above principles would be able to undertake dynamically the required transformation, before and during interaction, so as to provide the appropriate interactive behaviour for each user category to accomplish a given task. Thus, context-sensitive processing is the property that allows an MME to process alternatives and instantiate the one that is maximally preferred, given certain constraints.

3.3.2. Fusion: supporting context-oriented adaptive control: Assume that an MME integrates interaction facilities from three distinct design languages: the typical Web-like interaction, the book-like metaphor and the interaction facilities offered by a visualization tool. The book-like metaphor implements a top-level container such as the example of figure 6.

Being a top-level container, the book’s page can host a variety of interaction facilities, including elements of the other two design languages. In such a scenario, context-sensitive processing is needed to perform much more than making a choice amongst competing alternatives. Instead, it should devise a suitable mix of interaction facilities across the three design vocabularies. An example of such a mix is shown in figure 7. This type of adaptive behaviour is important when

Figure 6. A book-like top level container.
designing user interfaces to large information spaces where alternative interaction styles and techniques need to be mixed to facilitate effective, efficient and satisfactory interaction for different user groups.

The example in figure 7 illustrates a hypothetical user interface providing data on frequency of citation of research articles within a particular domain (e.g. activity theory). The overall embodiment of the interactive application complies with the book metaphor; it uses concepts such as book chapter and page, as container object classes, and next/previous page, table of contents, author index, subject index, etc., as navigation instruments. The left-hand page contains an HTML container with links to different thematic topics of an indexing system to a Digital Library. The right-hand page illustrates a visualization representing (1) collections of digital objects related to the thematic topic and (2) the popularity of digital documents by means of inter-referencing. In this manner, the user can identify popular articles in the Digital Library by considering how they are being referenced as well as when a particular theme was first published.

Such a user interface combines three radically different containers: the book that characterizes the overall interactive embodiment, the HTML page container with the links and the container of the interactive visualization graphic on the right hand page. The conditions for it to be built are higher-order than those of adaptability described above. Indeed, the underlying toolkits implementing the three distinct design languages should be fully interoperable, a property that is not fully implemented in presently available user interface development environments.

3.3.3. Implementing context-sensitive processing: From the above, it follows that adaptation in MME has a wider scope than the conventional forms encountered in the recent HCI literature. This has implications for the implementation of the context-sensitive processing layer of an MME. Specifically, the need to develop a new interactive embodiment introduces the additional burden of having to accommodate potential mismatches in the underlying programming models of the

![Figure 7. Mixing (fusing) metaphors as an example of adaptive behaviour of a Multiple Metaphor Environments.](image)
corresponding development toolkits. The issue is further complicated when interoperability of such development toolkits is necessary. To address the need for adaptive behaviours through metaphor fusion, an MME must implement appropriate abstraction mechanisms that will bridge across alternate development tools and offer a unifying development environment. One approach for achieving this goal is the architectural model shown in figure 8.

Figure 8 illustrates the intermediate layers that may be used to bridge across a machine- and user-oriented vocabulary. Specifically, each development toolkit is accessed through a dedicated component (e.g. toolkit server) that ‘knows’ how to map an abstract component specified in terms of unified object classes to toolkit-specific objects and behaviours. In this manner, access to toolkit resources is realized by linking to it rather than directly calling the toolkit. This is important when several toolkits may be concurrently active to realize alternative metaphoric manifestations for an abstract interaction component. Finally, such an integration of alternative and potentially distinct platforms need not be a programming-intensive task. In fact, there have proposals that address this issue in a generic manner. For example, in Savidis et al. (1997a), a Platform Integration Module (PIM) is described that generates a unifying programming model over two distinct toolkits. Using the output of PIM, developers can employ the same programming notation to call libraries and
functions of each one of the integrated toolkits, irrespective of their corresponding programming model.

4. Case studies on the articulation of design languages

4.1. Vertical expansion and its implications on a GUI

To illustrate how vertical expansion may be accomplished, we refer to a design case study related to accessing interactive applications built with a windowing system, such as for example Microsoft Windows, by disabled users and, in particular, those with motor impairments. This is a typical example of a design problem in which the design language (e.g. object classes, guidelines) supported by the original interactive environment does not suffice to accommodate the interaction requirements of motor impaired users. In early versions of these toolkits, motor-impaired users could not access the range of the facilities implemented. For purposes of illustration, we refer to two such examples: window management and text editing. Window management refers to moving and resizing windows on the desktop, minimizing/maximizing windows, etc. All these are typically implemented as mouse-based interactions, requiring fine temporal control. Impaired users with gross temporal control are instead more familiar and capable to cope with switch-based access.

One possible solution is to augment the basic object classes of the toolkit to enable additional interaction techniques suitable for the target user population. One such interaction technique, which is known to be effective for motor impaired users, is scanning. Scanning is a technique that allows the user to interact with the computer by selecting options from a target selection set using one or more switches. In order for an interface to be accessible through scanning, its interactive elements (e.g. top-level windows, interaction objects within them, etc.) need to be scanned either manually or automatically for the user to obtain access.

Augmenting the object libraries of a windows-like toolkit to facilitate the above interactive behaviour may be realized through subclassing, which allows scanning to become an integral property of the overall interactive environment. The resulting interactive behaviour should allow a disabled user to accomplish tasks (such as, for example, text editing), using dedicated interaction techniques. In the example of figure 9, two instances of vertical expansion are shown. The toolbars at the top illustrate an enhancement of the window management facilities so as to allow a user to carry out the designated window management tasks through explicit function activation (rather than the implicit function mode offered by the original toolkit). The technique involves scanning (either manual or automatic) of interaction objects and selecting the required option. The second example at the bottom depicts a virtual keyboard.

This new artefact automatically pops up when the focus is on a text entry field. Manual or automatic scanning is also in this case the primary means for navigating within the artefact to accomplish text editing. For more details on these examples and on the dialogue that implements the augmented environment, see Savidis et al. (1997b) for the case of Microsoft Windows.

What is important is that the augmented interaction facilities become an integral part of the overall environment and consequently developers may reuse them across different applications without expending additional resources. This should be contrasted against current practice, in which scanning is typically supported through programming, which increases the cost of development and maintenance of the user interface while also limiting reusability across applications.
4.2. Integrating a new design language

The alternative to vertical expansion is to use a design language to convey a particular interactive embodiment by means of introducing new top-level container object classes. This case can be realized through the development of new interaction

Figure 9. An example of augmenting a design language with new elements.
toolkits. As already mentioned, this frequently necessitates the development of new top-level containers and potentially distinct interaction mechanisms. As an example of this type of design language articulation, we will briefly discuss an example inspired from the non-visual implementation of the Rooms metaphor (Savidis and Stephanidis 1998), as realized in the COMONKIT non-visual interface development library.

COMONKIT was designed and implemented to facilitate the development of applications directly accessible by people with visual impairments who cannot articulate the visual modality to 'perceive' and interact with symbols in a conventional graphical desktop. For this purpose, it offers to the developer of the non-visual application 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., and a choice of input/output facilities. (See Savidis and Stephanidis (1998) for details of the authors' work and the applications built with it.)

Figure 10 shows an extract of a blind user’s interaction with an application built using COMONKIT. The user has indicated the intention to enter the communications room (to send an e-mail). The system provides various types of feedback upon entry and in the course of user’s interaction with the Room object.

The above example indicates that the non-visual interactive manifestation of sending an e-mail is radically different from its corresponding visual counterpart, even though the task performed by the machine is exactly the same. Nevertheless, as illustrated by Savidis and Stephanidis (1998), both can be available on the same machine and more importantly they can be executed concurrently.

5. Design guidelines for MMEs
The notion of an MME is grounded in the fact that no single interface implementation is likely to suffice for all different users and contexts of use. Thus, systems should be capable of exhibiting context-sensitive processing to facilitate the selection of alternative interactive behaviours suitable for different users and contexts of use. The question now posed is how can designers build systems

<table>
<thead>
<tr>
<th>Partner</th>
<th>Interaction</th>
<th>Object/Attribute</th>
<th>Modality</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>“ROOM IDENTIFICATION” sound</td>
<td>Room.EnterMSG</td>
<td>Audio sound</td>
</tr>
<tr>
<td></td>
<td>Welcome to Communications Room</td>
<td>Room.Label</td>
<td>Speech</td>
</tr>
<tr>
<td></td>
<td>You are on the floor of the room</td>
<td>TextReviewer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Choose one of the following options. Press R on the keyboard to review options. Press A on the keyboard to start again. Press E on the keyboard to exit the room.</td>
<td>Menu</td>
<td>Speech</td>
</tr>
<tr>
<td>User</td>
<td>Presses R on the keyboard</td>
<td>Command</td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>Three applications are available from this room. Press X on the keyboard to run application_1. Press Y on the keyboard to run application_2...</td>
<td>Menu</td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>“Application_1” speech</td>
<td>Command</td>
<td>Speech</td>
</tr>
</tbody>
</table>

Figure 10. Extract of non-visual interaction following the Rooms vocabulary.
exhibiting the properties of an MME. To this effect, this section consolidates practice and experience from several design case studies into three process-oriented design guidelines (i.e. **enumeration** of design alternatives, **encapsulation** of design alternatives into abstractions and **rationalization** of the resulting design space) and presents various techniques that could be used to bring them into effect.

Figure 11 shows the three guidelines and identifies techniques that may be used to accomplish each one. However, before each guideline is elaborated, it is perhaps appropriate to review some of their important characteristics that distinguish them from other design principles, such as those of user-centred design. In this context, our intention is not to invalidate user-centred design, which is considered as a basis, but instead to reveal how user-centred design can be enhanced and enriched through the proposed guidelines.

The guidelines for constructing MME are distinctively characterized by three attributes.

- They depict process-oriented recommendations.
- They inherit and bring forward an analytical perspective into HCI design.
- They facilitate the seamless integration of design and evaluation as components of a single inquiry.

The qualification of the proposed guidelines as process-oriented recommendations implies that these guidelines aim to provide designers with a principled approach, rather than principles *per se*, for attaining specific design goals. Thus, the guidelines provide designers with ‘know-why’ and ‘how-to-do-it’ guidance rather than ‘know-what’ and ‘what-to-do’ type of assistance. This is important, as much of the current work in HCI falls within the latter track, emphasizing specific techniques, which may or may not be of global applicability.

The second attribute of the guidelines is their analytical flavour. This entails a unifying and prescriptive framework for the conduct of design, which however, may accommodate task- or situation-specific approaches (e.g. predictive usability evaluation by means of a cognitive model, application of software ergonomics criteria, heuristics, etc.). However, what is important in this context is not the specific

---

![Figure 11](image-url)  
**Figure 11.** Requirements for designing for the broadest possible end user population.
technique that a designer may choose to employ, but the deliverables that should be compiled at the end of the design inquiry. Thus, the proposed guidelines do not describe what is to be done; instead they prescribe what is to be achieved through analysis.

The final attribute of the proposed guidelines is their holistic notion of design as a scientific inquiry, which blends engineering and evaluation. Traditionally, the field of HCI used to distinguish between what is actual design work and what constitutes evaluation. This resulted in the separation of expertise and responsibilities, which may be the primary reason why industry is still far from responding favourably to usability engineering. The proposed guidelines adopt a slightly different viewpoint. Evaluation is not intended to provide a means for refining designs, but a means for informing and rationalizing design spaces. Thus, irrespective of who conducts the evaluation or how it is operationally carried out, it is a necessary condition to complete the design cycle. Phrased in another way, the role of evaluation is not only to provide feedback, as in user-centred design, but also to generate design ideas and creative concepts. It follows, therefore, that without evaluation as a mandatory design aid, the proposed guidelines cannot be fully followed as the designer is likely to come across decision points that can only be addressed through some sort of evaluation. This is substantially different from the view that currently prevails and which considers evaluation as a complementary stage offering a tight evaluation feedback loop in the context of user-centred design.

5.1. Enumeration of plausible design alternatives
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 abilities, requirements and preferences, tentative designs should aim to accommodate the broadest possible range of capabilities across different contexts of use. To ensure this, designers should employ techniques that allow them to reach as broad a range of the target user population as possible, in order to identify boundary conditions and failure scenarios. It follows, therefore, that instead of restricting the design activity to producing a single ‘good-enough’ outcome which is made available to users for review, designers should strive to compile design spaces containing plausible alternatives.

As an example, consider the primitive interaction task of selection. A selection may be made either by choosing an option from a menu or by issuing a command, etc. Moreover, a menu may be conveyed in different design languages, such as the restaurant, the typewriter or electric devices (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 considered suitable for a blind user who lacks the capabilities to attain information conveyed in the visual modality. Instead, blind people would be more comfortable using alternative manifestations conveyed either in audio (e.g. synthetic speech) or tactile modalities (e.g. a tactile listbox).

5.2. Encapsulation of plausible alternatives into design abstractions
Once the design space has been compiled and documented, the design activity should proceed towards the encapsulation of plausible alternatives into abstract, reusable
and extensible design components. The need for abstraction is twofold. First, abstractions may be used to decouple a design concept from any particular physical realization which is tied to a specific interaction platform; thus, through abstractions, a design concept may be mapped onto alternative physical counterparts, thus becoming tailor-made for new design problems.

Second, abstractions provide a mechanism to support incremental design and design updates, as requirements mature, or evolve; in this manner, the original design space may be extended to include new physical realizations, necessitated by new contexts of use or made possible by novel interaction technologies. Consider, for example, the task of selecting within a container object class. Removing the presentation-specific details, which differentiate the interactive behaviour of the container (in a visual, auditory or tactile modality), one arrives at a minimal set of attributes that constitute an abstract container, and which can capture the global execution context of selection. For example, an abstract container can be specified in terms of the attributes shown in table 1 (ACCESS Consortium 1995).

The above attributes depict an abstract container, which is independent of user characteristics, platforms or contexts of use. Instantiating the attributes of the container, we can derive concrete manifestations of such containers which are implemented in a target platform. For instance, the virtual keyboard of figure 9(b) could be specified using the scheme of table 2 (Akoumianakis and Stephanidis 1999). In table 2, the virtual keyboard is specified as a collection of one container (line 1) and button object classes (line 22). Object classes such as buttons have attributes, which are represented by the predicate:

\[
\text{Attribute}(\text{symbol, AttrDom}), \hspace{1cm} \text{AttrDom is (symbol, string, listOfParameters)}
\]

Thus, each object category is modelled as a collection of attributes. In the specification of table 2, lines 2 – 21 declare attributes for the container. Each attribute may be associated with specific parameters, which define exactly how the attribute is supported by various user interface development systems. For example, lines 6 – 19 declare the parameters of scanning in the container. Attributes are typically classified into three different categories: general (lines 2 – 4 and 20 in table 2), presentation (lines 23 – 24 in table 2) and behavioural attributes. General attributes are possessed by all interaction object classes and include non-trivial assignments such as (input/output device, interaction techniques, feedback, and for container object classes, navigation, access, topology and access policies. Presentation attributes detail the

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access.Policy</td>
<td>Policy for accessing and obtaining focus on the container</td>
</tr>
<tr>
<td>Topology.Policy</td>
<td>Policy for organization objects contained</td>
</tr>
<tr>
<td>Navigation.Policy</td>
<td>Policy from moving from one object to another within the container</td>
</tr>
<tr>
<td>Initiation.Feedback</td>
<td>User feedback conveyed upon start of interaction with the container</td>
</tr>
<tr>
<td>Interim.Feedback</td>
<td>User feedback while interacting objects within the container</td>
</tr>
<tr>
<td>Completion.Feedback</td>
<td>User feedback conveyed upon end of interaction with the container</td>
</tr>
<tr>
<td>Metaphor</td>
<td>Identifier depicting the interaction platform implementing the container</td>
</tr>
</tbody>
</table>
5.3. Rationalization

Rationalization of the design space entails a principled approach to defining the reasoning behind each alternative. A rationalized design space enables the designer to make informed decisions as to how design abstractions can be mapped to suitable physical counterparts. Additionally, it provides evidence for the mapping. To produce the necessary evidence, designers may have to assess alternatives with end users, or carry out experimentation to develop comparative results and decide on maximally preferred options. Thus, for example, a comparative test may be set up to assess the circumstances and the parameters determining alternate choices for making a selection. Such a comparative test may be carried out using alternative user-centred design techniques, including subjective user assessments, performance measurement, GOMS analysis, etc. It follows therefore that mapping design abstractions to concrete alternatives is a highly situated activity, as alternative options may be necessitated by either different users groups or different contexts of use.

The need for rationalization necessitates a shift from artefact-oriented design towards process-oriented and analytical design, whereby the reasoning behind an artefact is revealed, articulated and documented so that it can be subsequently retrieved, reviewed and/or modified. In the remaining of this section, we will briefly

| 1. | [ container,          |
| 2. |   [ attribute(general,(accessPolicy, keyboardEmulator, [])),   |
| 3. |     attribute(general,(topologyPolicy, horizontal, [])),   |
| 4. |     attribute(general,(inputDevice, twoSwitchScan, [])),   |
| 5. |     attribute(general,(inputTechnique, scanning2D,   |
| 6. |       [ parameter(selectionSet, symbol, []),   |
| 7. |         parameter(scanmode, 1, []),   |
| 8. |         parameter(timeScan, 100, []),   |
| 9. |         parameter(dm_bordercolor.red, 1, []),   |
|10. |         parameter(dm_bordercolor.green, 1, []),   |
|11. |         parameter(dm_bordercolor.blue, 1, []),   |
|12. |         parameter(dm_borderstyle, 2, []),   |
|13. |         parameter(dm_borderwidth, 1, []),   |
|14. |         parameter(em_borderwidth, 3, []),   |
|15. |         parameter(em_bordercolor.red, 1, []),   |
|16. |         parameter(em_bordercolor.green, 10, []),   |
|17. |         parameter(em_bordercolor.blue, 255, []),   |
|18. |         parameter(em_borderstyle, 1, [])   |
|19. |   ]),   |
|20. |   attribute(general,(outputDevice, on_Screen, []))   |
|21. |   ...   |
|22. |   ]   |
|23. | button,   |
|24. |   [ attribute(presentation,(label.font, helvetica, [])),   |
|25. |     attribute(presentation,(label.size, 12, [])),   |
|26. |     ...   |
|27. |   ],...   |
examine how rationalization can be approached to provide a sound basis for selecting maximally preferred options within a design space. First, however, we formulate the rationalization problem in general terms.

Let $X$ be a design variable that is to be decided by selecting options from $S = \{S_1, S_2, \ldots, S_n\}$. Rationalization depicts the process whereby a decision is made using some sort of technique, which provides reasonable justification for the choice. It should be noted that making such a decision, may be arbitrary or the result of a reasoning process. We are interested in the latter case, which assumes that rationalization is a knowledge-intensive exercise. Thus, we are interested to identify the knowledge items needed, and the reasoning mechanism, which can deliver a solution. For this purpose, it is assumed that rationalization involves assessing alternatives against a designated set of criteria $C$, in order to select what is optimal for $X$, given the current state of the knowledge available.

Formally, rationalization entails four steps.

1. Identifying $X$.
2. Enumerating $S$.
3. Specifying the set of relevant criteria $C$.
4. Devising a technique which allows the decision-maker (e.g. the human designer or a computer program) to select for $X$ a maximally preferred option from $S$, given $C_1 \in C$.

The recent literature on HCI, and particularly the part concerned with intelligent user interface technologies, report various formal, or semiformal techniques which have been used to attain the last step in the rationalization process. Some of these make use of mathematical models, such as Markov chains analysis, queuing models and utility models. Others rely on articulating accumulated wisdom, documented in guidelines or style guides, into inference rules, while yet another cluster involves specialized techniques geared towards the generation of empirical evidence to inform design by means of experimental set ups (e.g. comparative experiments) or analytical/engineering models such as GOMS.

For the purposes of this paper, we will briefly elaborate on an alternative approach to rationalization, which is based on social choice theory for making joint decisions and resolving conflicts. The basic premises of the technique were described in a previous article (Stephanidis and Akoumianakis 1997). In addition, a system implementing the technique was described in Akoumianakis and Stephanidis (1999). In this section, we generalize previous efforts and extend the scope of their applicability.

Thus, for the purposes of rationalizing design spaces to suit context-sensitive processing by MMEs, we formulate the rationalization problem as follows.

Given a design issue $I$, a set of application-specific task contexts $TC$, a set of designated criteria $C$ and a set of design alternatives $O$, then:

$I$: Dependent variable
$O$: Factor level of the dependent variable
$C$: Independent variable
$K$: Preference expression over the elements of $O$ for a particular $TC_i$ and $C_i$ and can be either a preference or and indifference:
Given the above, rationalization entails deriving a preference ordering over $O$. A preference ordering is a ranking of equivalent classes of alternatives with respect to a particular design criterion. These equivalent classes are also called indifference classes, since an agent (i.e. the system, the designer or the user) prefers alternatives from one class to those from another. That is, one indifference class ranks above another, if its members are preferred to those of the other class. We adopt the notation $\{x, \ldots, y\}$ to represent indifference classes. Thus, $\{x, y\}$ means that $x$ and $y$ are of the same equivalent class with respect to a criterion $C$.

In conclusion, it is important to mention that the above scheme may be used to rationalize a wide range of design issues, at different levels, including:

- semantic options (e.g. choice of interactive metaphor);
- syntactic properties of a dialogue (i.e. function activation mode, guidance); and
- physical attributes of interaction object classes (i.e. input/output devices, interaction techniques, feedback modalities, etc.).

6. Discussion: implications of MMEs and future research

We have detailed the principles underpinning and the practice involved in the design and development of MME. Here, we shall briefly review some of the implications of MME on prevalent HCI design and development conceptions and identify areas of future work. Our intention will be threefold: (1) to contrast the underlying notion of an MME against contemporary views on interactive systems; (2) accordingly to explain how the propositions discussed in this paper suit better the study of diversity in the context of a human-centred Information Society; and (3) to examine some prominent direction for future research. We intend to do this by elaborating on the design and development implications of MME and, in particular, the two dimensions that distinctively characterize such systems.

6.1. Capturing the global execution contexts of tasks

A first point of discussion relates to the scope of the design activity. Traditionally, designers were confronted with the task of devising the elements of an interactive computational embodiment, which would enable users to carry out certain activities. To respond to diversity, their works frequently draw upon theory, or consolidated knowledge (e.g. guidelines) to inform design. However, both these tracks have been suboptimal, resulting in weak conceptions and accounts of diversity in HCI. In particular, the theoretical strand has been of partial influence on HCI design, particularly with regards to generating innovative products and services. This has led to critics about the sufficiency of the theoretical links of HCI with information processing psychology. The literature reports debates that point to the need for either extending the type and scope of information processing psychology, or drawing upon more developmental sciences to derive theories which capture and explain a wider set of interactive phenomena.

On the other hand, use of consolidated knowledge and best practice as an approach to study diversity introduces a reflective paradigm for HCI design. This may be sufficient for identifying shortcomings and design flaws but cannot drive
innovative product development beyond contemporary principles. The problem has been even more complicated with recent technological developments and the proliferation of Internet technologies. Today, it is widely accepted that diversity in the context of HCI is no longer a strictly human issue. In addition to variety in human capabilities, there is variety in technological options, as well as variety in contexts of use.

Consequently, any system aiming to address diverse requirements needs to accommodate the global execution context of the tasks it facilitates. In other words, designers increasingly need to engage in argumentative design to envision potential breakdowns, identify plausible solutions and, accordingly, devise alternatives which may suit a wider range of users, technological platforms and contexts of use. This paper has described how this can be attained through structured and iterative cycles of enumeration, encapsulation and rationalization. Of particular interest is the compelling need for capturing the reasoning behind alternative design options in order to provide a mechanism for implementing context-sensitive processing.

6.2. Specifying versus programming interactions
The second prominent issue, which relates directly to capturing the global task execution context, is the implementation of MME. Clearly, the current generation of tools have a limited scope which is typically bound to a particular (or a few in the case of multiplatform toolkits) interaction platform, and a particular class of interaction elements (e.g. buttons, menus, windows). On the other hand, MME assume a strong notion of abstraction beyond metaphor or corresponding platform.

Our experience indicates that such systems cannot be realized by conventional user interface programming tools, since the latter are tightly coupled to low level platform details. What is really needed is specification-oriented environments that embody the capability to negotiate with different target platforms and toolkits in order to decide how a specified abstract artefact is to be manifested interactively. For this to be realized, the specifications should not make direct calls to the platform but rather link to it to make use of available interaction resources. This, however, raises a whole new range of issues on how to realize such ‘linking’. One approach towards this end is the unified development methodology (Stephanidis 2001), which provides the architectural abstraction and underlying mechanisms to generate interactive code from specifications.

7. Summary and conclusions
This paper has presented an engineering perspective that provides a foundation for building MME. These are integrated systems capable of exhibiting context-sensitive processing to map functions of a target domain to symbols in a suitable source domain, and vice versa. We have presented the principles underpinning such systems, guidelines for their design, techniques that can be employed and development requirements needed to realize systems exhibiting the properties of MME.

The conclusions drawn from this work can be summarized as follows. First, MMEs provide a rich framework for studying diversity and its implications on HCI. Specifically, they foster a conception of the computer as a mediational device for carrying out context-oriented human activities. The notion of context-sensitive processing in MMEs offers a mechanism for capturing and reasoning about computational transformations that meet contextual requirements. It is precisely this capability, however, that renders MMEs a suitable frame of reference for designing
and developing interactions for the broadest possible end-user population, thus supporting and facilitating a tractable perspective on design for all in HCI.

Second, though orthogonal to prevalent design techniques, MMEs require a broader methodological frame of reference than the prevailing artefact-oriented practices in HCI. This is due to the inherent focus on understanding diverse user groups and contexts of use to facilitate suitable adaptations. Note that adaptation in this context not only implies mere lexical or syntactic modifications of interactive behaviour, but also radical departures from a particular computational embodiment, such as that of the desktop computer, to facilitate computer-mediated human activities in a context. For this to be possible, designers need proactively to identify and reason about context and develop specifications of interactive behaviours, which are suited to certain conditions. In other words, they increasingly need to envision potential breakdowns, unconventional use and situated action.

Finally, MMEs require new and a broader range of tools to facilitate their development. Specifically, tools are needed to implement alternative interactive manifestations for an artefact. As already discussed, this may involve horizontal or vertical expansions of a development platform, which has proven to be a non-trivial task. Especially the case of horizontal expansion may necessitate use of distinct or dedicated development toolkits, which will need to be integrated with other toolkits with regards to underlying programming models, APIs, etc. We demonstrated previously how such integration can be attained and described the challenge involved (Savidis et al. 1997a). Moreover, the development of such tools should be accompanied with a shift in prevailing practices from programming interactions to specification-based frameworks. The current generation of tools, with only a few exceptions (e.g. model-based approaches, 4GL-based user interface management systems) offer limited support for this purpose.

Acknowledgment

To Gavriel Salvendy, a source of energy and inspiration.

References


AKOUMIANAKIS, D., Savidis, A. and STEPHANIDIS, C. 2000, Encapsulating intelligent interactive behaviour in unified user interface artefacts, Interacting with Computers [Special Issue on The Realities of Intelligent Interface Technology], 12, 383–408.


