The HOMER UIMS for dual user interface development: Fusing visual and non-visual interactions

Anthony Savidis, Constantine Stephanidis*

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

Received 30 June 1997; received in revised form 28 October 1997; accepted 26 March 1998

Abstract

Existing systems which enable accessibility to graphical user interfaces (GUIs) by blind people follow an 'adaptation strategy': each system adopts its own hard-coded policy for reproducing visual dialogues in a non-visual form, without knowledge about the application domain or the particular dialogue characteristics. It is argued that non-visual user interfaces should be more than automatically generated adaptations of visual dialogues. Tools are required to facilitate purposeful non-visual interface construction, allowing iterative design and implementation. Such tools should cater for the construction of 'integrated' user interfaces, which are concurrently accessible by sighted and blind users. Thus, the concept of dual user interfaces is introduced, arguably as the most appropriate basis to address this important issue of concurrent accessibility, in order to prevent segregation of blind people in computer-based working environments. A user interface management system (UIMS) has been developed, called HOMER, which facilitates the development of dual user interfaces. HOMER supports the integration of visual and non-visual toolkits of interaction elements; a non-visual toolkit, called COMONKIT, has been also implemented for building non-visual user interfaces, and has been incorporated in HOMER. © 1998 Elsevier Science B.V. All rights reserved

Keywords: Dual interface; Non-visual interaction; Abstract interaction objects; User interface specification; User interface management systems
1. Introduction

1.1. Background and rationale

Currently, access to graphical user interfaces (GUIs) by blind users is enabled through systems which reproduce the lexical structure of user interfaces (i.e. interaction objects and their relationships) to a non-visual form. These are characterized as adaptation oriented methodologies and the relevant systems, which employ filtering techniques to extract lexical information, are commonly referred to as screen readers. One of the main weaknesses of many screen reader programs is that they explicitly introduce visually oriented concepts to the non-visual interaction. Such concepts come from the spatial metaphor, which was developed as a result of intense research efforts based on the concept of exploiting the immense human visual information processing capabilities. Recent adaptation approaches aimed at excluding specific types of objects from reproduction into a non-visual form and employed 3D audio output techniques for non-visual layout construction [1]. However, such lexical level filtering and subsequent lexical transformation of the user interface suffer from many significant theoretical and practical drawbacks (see Fig. 1).

First, the automatic ‘transformation’ of visual interfaces to non-visual interactions is based merely on run-time lexical information (and interaction object hierarchies). This is because the syntactic and semantic designs are directly mapped in ‘code’, during the transition from the design stage to the implementation stage, from which filtering of syntactic or semantic aspects is not possible. In addition, the object hierarchies extracted at run-time reflect only the implementation ‘picture’ (i.e. not the original lexical design) of the visual lexical design (if programmed differently then a different object structure would probably be extracted); they simply ‘tell’ part of the lexical design ‘story’ (i.e. visual effects, graphical illustrations, iconic design, etc. are not extractable at run-time).

Fig. 1. The elimination of non-visual interaction design and implementation stages in screen readers.
Second, this restricted view of the visual lexical design for visual applications, as it is automatically extracted at run-time, is defined on the basis of sighted user needs, following the semantic and syntactic design phases, in the overall interface design process. Screen readers perform an automatic translation of the object hierarchies to a non-visual form without any knowledge about the particular semantic and syntactic context; this is similar to translating text from one language to another by simple substitution of words. Clearly, the quality of the artefact produced, following such a procedure, is questionable.

Finally, considering the current trend of user interface software and technology (UIST) towards highly interactive visual methods, such as virtual reality and 3D representations (e.g. visual reality, such as the perspective wall and cone trees [2] and WebBook/Web-Forager [3]), and complex 2D visualization techniques [4], it is likely that automatic adaptation techniques, based on filtering and reproduction, will become unrealistic or meaningless in the future due to (a) implementation barriers (i.e. interface elements not extractable) and/or (b) poor interaction quality (i.e. the inherent complexity of software applications will likely pose the need for an explicit non-visual interface design process).

Even with the application of very sophisticated run-time filtering, in situations where information is conveyed in various pictorial forms (i.e. images, tabular data representations, bar-charts, diagrams, illustrations, artistic scenes, etc.), its automatic reproduction to a non-visual form is not feasible, since no extractable data (i.e. data that can be filtered) which concern the represented semantics, are stored internally. Currently, it is not possible to achieve practically automatic deduction of the internal semantics, exclusively from pure geometrical descriptions (i.e. image understanding methods). Moreover, domain-oriented knowledge is required in order to distinguish whether common symbolisms correspond to differentiated semantics (i.e. resolving presentational ambiguity). Today, the widespread use of the World Wide Web as a primary means of information provision and exchange, in which graphical illustrations are extensively used, imposes the necessity for providing alternative representations via high-quality non-visual dialogues.

We have defined the concept of dual user interfaces, reflecting a fundamental set of interaction requirements targeted towards equal user interaction support for blind and sighted people. In this context, we distinguish the following technical issues as being of key importance for developing dual interfaces: (i) support for shared specification of subdialogues by capturing syntactic commonalities (when this is practically possible); (ii) employment of abstract interaction objects; (iii) loose syntax dialogue control methods; and (iv) ability to utilize concurrently interaction elements from two toolkits (a visual and a non-visual one) for interface implementation.

We have considered a language-based development framework as appropriate to meet these requirements, and we have developed the HOMER user interface management system (UIMS), providing the HOMER language which has been specifically designed for dual interface development. We will discuss the various dialogue characteristics of dual interfaces, followed by the identification of key development requirements relevant to dual interaction. The discussion of the development features supported by the HOMER UIMS will be carried out on the basis of an interactive application, supporting dual hierarchical picture exploration. The dual application scenario will be incrementally constructed, on the one hand revealing the most important development needs in dual interface development, while on the other hand demonstrating how those requirements are
effectively met through the development mechanisms of the HOMER UIMS. In this process, we will also justify: (a) the decision to support a language-based development approach of dual interfaces; (b) the value of the various special-purpose features of the HOMER language for dual interaction development; and (c) the innovation of those development features, in comparison to existing development tools.

1.2. The concept of dual interaction

A dual user interface is characterized by the following properties: (i) it is concurrently accessible by blind and sighted users; (ii) the visual and non-visual metaphors of interaction meet the specific needs of sighted and blind users respectively (they may differ, if required); (iii) the visual and non-visual syntactic and lexical structure meet the specific needs of sighted and blind users respectively (they may differ, if required); (iv) at any point in time, the same internal (semantic) functionality should be made accessible to both user groups through the corresponding visual and non-visual ‘faces’ of the dual user interface; (v) at any point in time, the same semantic information should be made accessible through the visual and non-visual ‘faces’ of the dual user interface respectively.

The first requirement is important for enabling cooperation between blind and sighted users and thus avoiding further segregation of blind people in their working environment. The second requirement explicitly poses the need for interaction metaphors specifically designed for blind users. The third requirement expresses the need for separate non-visual interface design. Finally, the fourth and fifth requirements indicate that the underlying functionality and information structures should be made equivalently accessible, though via different interactive channels.

1.2.1. An example of dual interaction

In Fig. 2, a simple dual interface is illustrated depicting hierarchical structures (left) and numerical values (right). The visual appearance employs 3D graphics and highly visual symbolisms, while the non-visual dialogue may be structured in a radically different manner and may combine Braille, speech and 3D sounds for output, while 3D pointing and speech may be used for input.

This example is also indicative of the need for having distinct syntactic and lexical designs for the visual and non-visual dialogues. More specifically, the visual interface (see Fig. 2) employs information visualization for presenting hierarchical structures by means of interactive graphical trees; facilities such as dragging, pop-up information boxes, and animation are employed, clearly being inaccessible to blind users. Hence, the visual metaphor (i.e. visual tree structure) and its respective dialogue design become inappropriate for realizing the (concurrently available) blind-user interaction. This last remark reveals the necessity of separately designing a non-visual interface, which, as illustrated in Fig. 2, employs a 3D-auditory, 3D-pointing interactive realization of hierarchical information structures (such dialogue methods are described in Ref. [5]), together with Braille output techniques. This scenario has two important features: (a) the visual interface, providing typical direct manipulation visualization techniques, cannot be ‘filtered’ by screen readers (only the interaction objects supplied by the target interaction platform
can be filtered); and (b) the visual interaction design requires such radical refinement that it practically leads to a dedicated non-visual interface design.

1.2.2. Dimensions of blind–sighted user cooperation in dual interaction

One of the basic requirements of dual user interfaces is to facilitate concurrent access by blind and sighted users (i.e. a type of collaboration). In this context, the driving objective of concurrent access in dual interaction is fundamentally different from the goals of cooperation in traditional computer supported cooperative work (CSCW) applications. In the latter, users have to perform a single job, all contributing separately towards its accomplishment in a shared workspace; hence, there is a mainly task-oriented employment of CSCW techniques in an organizational context, aiming primarily towards process improvement. However, in the former case, concurrent access is facilitated so as to avoid segregation of blind users in the working environment. Imagine a blind user in an enterprise context, performing some computerized work tasks, like data input/retrieval in/from a database. Then it is desirable to support the scenario in which a sighted worker approaches the terminal with which the blind user is working, being able to see a visual picture of the running application(s), while also having the capability to directly perform visual interaction. Without such an establishment, initiation of cooperation, which requires some typical conversations such as ‘let me show you this’, ‘I think you should change that’, or ‘can we search about that?’ etc., cannot be facilitated, unless both users have an appropriate interactive picture of the subject items. As we will discuss later, this is a type...
of local collaboration, not involving computer-mediated cooperation properties; we have also supported the basic type of remote collaboration, involving a sighted and a blind-user working on the same dual application from different machines. There are a number of significant dimensions which concern concurrency of interaction with respect to the visual and non-visual ‘faces’ of a dual user interface (the ‘faces’ are what a blind and a sighted user perceive as their respective interface), as shown in Table 1.

The A dimension is related to concepts employed for communication between the two user groups (i.e. discussion during collaboration) and can be either lexical (e.g. ‘this menu, this button’), syntactic (e.g. ‘I am doing editing’) and semantic (e.g. ‘the recording volume has changed’).

The B dimension concerns the metaphors of interaction for the visual and non-visual environments which can be either identical (e.g. Desktop and Desktop) or different (e.g. Desktop for visual and Rooms for non-visual [6]).

The C dimension characterizes the freedom that blind and sighted users have on performing actions independently of each other. More specifically, the dialogue control can be either synchronous (i.e. the visual and the non-visual dialogues pass always from the same ‘states’ by progressing in strict parallelism), semi-synchronous (i.e. users have freedom to focus on different tasks, although for certain points synchronicity is imposed) and asynchronous (i.e. users may focus on different tasks, interacting on different interaction objects and performing different actions).

The D dimension characterizes the flexibility on the physical structure of the visual and non-visual faces of a dual user interface. It is possible to have identical structure (e.g. same object classes and instances, same layout, same object hierarchies), similar structure (e.g. some of the interface components and object classes are different, but the overall physical similarity is evident) and totally different structure (e.g. the hierarchical organization of objects is completely different involving different classes of objects—no physical similarity can be identified).

The E dimension concerns the flexibility of applying a different dialogue design for each environment. The dialogue structure can be either identical (e.g. same decomposition of user tasks) or different (e.g. different organization of tasks for blind and sighted users).

Finally, the F dimension concerns the type of collaboration which can be either local (i.e. users work on the same machine and are physically close to each other) or remote (i.e. users work on different machines—distant collaboration).

The properties related to the concurrency of interaction for dual user interfaces can be defined by any sextuple from the set \( A \times B \times C \times D \times E \times F \). In comparison, the
concurrency supported by adaptation-oriented approaches has properties which belong to the following set: $A \times \{\text{Identical}\} \times \{\text{Synchronous}\} \times \{\text{Identical, Similar}\} \times \{\text{Identical}\} \times \{\text{Local}\}$. It is evident that apart from the theoretical and practical drawbacks from which adaptation-oriented approaches suffer (as mentioned previously), the type of interactive collaboration with sighted users that can be supported is considerably restricted.

2. Related work

We will discuss related work under two perspectives. The first will concern work relevant to adaptation-oriented approaches for providing blind-user access to GUIs. The second deals with past work in the context of interface building techniques and methodologies, mainly user interface management systems (UIMS), judging their appropriateness with respect to the key technical issues previously identified for developing dual interactions.

2.1. Adaptation-oriented approaches for blind-user access to visual interaction

Solutions which appeared in the past for providing accessibility to graphical user interfaces by blind people were based on filtering techniques aiming to reproduce in a non-visual form an internally stored image of the visual display (off-screen model). Examples of well-known commercially available systems are: OUTSPOKEN® by Berkeley Systems Inc., and SYSTEM 3® by the TRACE Research and Development Centre at the University of Wisconsin, for the Macintosh; and SLIMWARE WINDOWS BRIDGE® by Syntha-Voice Computers Inc., for WINDOWS®.

Research has been also devoted to providing access to the X WINDOWING SYSTEM; two different approaches are discussed in Ref. [1]: (i) the one followed by the MERCATOR project at the Georgia Institute of Technology (USA), which transforms the user interface of X-clients to an appropriate auditory representation; (ii) the other, followed by the GUIB project, supported by the TIDE Programme of the European Commission (DG XIII), addresses both the WINDOWS® environment and the X WINDOWING SYSTEM. The GUIB approach is based on a proper transformation of the desktop metaphor to a non-visual version combining Braille, speech and non-speech audio. Additionally, other issues have been investigated by GUIB [7], including different input methods which can be used instead of the mouse, the problem of how blind users can locate efficiently the cursor on the screen, issues related to combining spatially localized sounds (both speech and non-speech) with tactile information and design of appropriate non-visual interaction metaphors. The GUIB project has also supported two different activity lines: the adaptation oriented line, under which the above efforts fall, and the future development tools line, in the context of which the work reported in this paper has been partially funded.

2.2. Interface construction techniques in the perspective of dual interaction

We will focus on two primary issues: (i) support for abstraction of interaction elements,
and (ii) ability to integrate toolkits (including non-GUI toolkits). In this context, past work falls into the following categories.

**Generalizing and reusing primitive input behaviour.** Various input behavioural aspects of interaction in direct manipulation graphical interfaces are captured, generalized and expressed via a set of generic highly parameterized primitives through which complex forms of behaviour can be assembled in a modular manner. Initial work in this field is mainly concerned with interaction tasks [8], providing a comprehensive theoretical framework, while interactors [9] provided an implementation-based framework for combining input behaviour to create complex interactions. Both addressed the desktop metaphor for graphical interaction and propose an alternative structural and behavioural ‘view’ of the lexical level; hence, they are not related to the abstraction domain.

**Visual construction with behaviour abstraction.** This approach has been realized initially in the application display generator (ADG) system [10], and is based on the visual construction of interfaces on the basis of behavioural abstraction of interaction objects; the developer may choose between different physical realizations of such abstract behaviour. Through different choices for physical realization of abstract objects, alternative versions of the lexical structure of the developed interface are produced. The abstractions supported in the ADG system are visual objects such as ‘bar gauges’, ‘scrollbars’, etc., which can have alternative representations. The number of such abstract objects, as well as the links between abstract objects and specific physical alternatives, are fixed in the ADG system. The previous type of ‘abstraction’ should be better called generalization, since the resulting object classes are still related to the desktop metaphor, merely providing a collection of generically applicable types of desktop behaviour.

**Integration of platforms.** In this case, interface tools initially provide no built-in lexical interaction elements (i.e. the tabula rasa concept), but they supply facilities for: (i) connecting the interface tool to a particular toolkit; and (ii) making the interaction elements of imported toolkits available through their supported interface construction methods. The SERPENT UIMS [11] is the first system to support platform integration. It provides a dedicated language for this purpose [12], apart from dialogue specification, in which all interaction elements are considred as objects. However, this integration model fails in the following situations: (a) apart from interaction elements, particular toolkit functionality should be made available to interface developers (e.g. toolkit resource management, drawing facilities), and (b) direct handling of device-oriented events is required for low-level dialogue control (modelling input events as interaction objects is problematic, due to their fundamentally temporal nature). Also, SERPENT does not enable an alternative toolkit to be concurrently connected (i.e. the necessary ‘hook’ for the non-visual toolkit is missing).

**Formal models of interaction objects.** Such models have been strongly related to methods for automatic verification of interactive systems, in which formal specification of the user interface is employed to assert certain properties; the theoretical background of such approaches is reactive systems from distributed computation theory. The most representative example is the interactor model [13,14]; interactors convey both state and behaviour, communicate with each other, as well as with the user and the underlying application. Such an entity category is computationally very primitive (as a result, highly generic), and is appropriate for modelling in detail the dialogue of objects from
within (i.e. the physical dialogue of an object as such). The interactor model itself constitutes an algorithmic computation structure, like state automata, and as a result is far away from typical programming models of interactive entities, like interaction objects in toolkits. Hence, interactors may be highly expressive to allow interaction objects to be formally represented as collections of communicating interactor instances; however, they are less convenient as a development entity in an interface tool (in the same manner that a Turing machine, though the most generic algorithmic structure, is employed as an explicit programming model in programming languages). In conclusion, the interactor model, though being highly generic, is better suited for formal verification approaches (and has been very successful in that respect), rather than interface development languages.

**Meta-widgets.** The concept of meta-widgets is based on the abstraction of interaction objects above physical platforms [15]. A meta-widget is free of physical attributes and presentation issues and is potentially able to model interaction objects above metaphors. Currently, meta-widgets have been provided with a fixed implementation [15], with respect to the classes of meta-widgets and their physical instantiation. It should be noted that the concept of virtual interaction objects in the HOMER language, as defined in Ref. [16], is similar to the notion of meta-widgets.

In conclusion, it was found that past work has not addressed the provision of interface development techniques for: (i) creating abstract interaction objects; (ii) defining multiple alternative schemes of mapping abstract objects to physical interaction objects (i.e. polymorphism); and (iii) selecting the desirable active mapping schemes for abstract object instances, in the context of interface implementation. Additionally, the ability to handle two toolkits, as is fundamentally required for dual interface implementation, is not supported. The absence of such development methods from existing tools is due to the different development needs of typical graphical interactive applications, with respect to dual interfaces. All the above technical issues become top priority in the context of dual interface construction. Moreover, we believe that the abstraction mechanisms of the HOMER language, which have been originally designed for dual interaction development, may also contribute towards a more efficient and effective organization of interface implementation layers in general-purpose interaction development.

### 2.3. Applications specifically designed for blind users

Such work emphasizes the importance of designing interactive applications by explicitly supporting a non-visual interface design process, in which blind-users’ needs are those which drive the design of the dialogue aspects. The Soundtrack system [17] has been an experimental word processor, providing an auditory interface for blind users, which was intended to be a vehicle facilitating research work, rather than providing just another word processor [18]. From such work, some key questions have been posed [18]: (i) the appropriateness of speech versus non-speech sounds; (ii) the accuracy of finger-based input; and (iii) considering that a pointing device is indeed important for blind users, it still remains to be assessed whether the mouse is appropriate for that role. The most representative examples of more recent work are Web browsers for the visually impaired, such as pwWebSpeak® from The Productivity Works, Inc., as well as more research-oriented systems such as V-Lynx [19].
3. Dual application development subject-case

We will provide a real-life dual interactive application example, briefly reported in Ref. [20], which will constitute the main development scenario throughout this paper. On the basis of this application scenario we will: (i) identify the key functional criteria for interface tools, in order to support dual interaction development; (ii) review existing tools, with respect to those specific criteria, and show that the required functionality is missing (from present systems); and (iii) describe the added-value features of the HOMER UIMS, which appropriately address the identified criteria, as they serve the distinctive development demands of our application scenario. We provide below a brief description of the application scenario.

We need to build a dual application which will employ the Athena (Xaw) widget set of the X Windowing System for the visual interaction, and the Rooms/COMONKIT toolkit [6] for the non-visual interaction. The application should facilitate interactive exploration of images, by supporting annotations on image regions with textual content; the dual exploration dialogue should work in a semi-synchronized fashion (i.e. when a blind user reviews a particular region, the visual interface presents the same region, and vice versa, though the region selection dialogue as such is not synchronized). Direct manipulation techniques for visual interaction will be implemented, enabling the user to directly review regions through the mouse. Additionally, menu-based interaction will be supported for both users, allowing hierarchical picture exploration; in this case, interaction is to be performed on the basis of a predefined hierarchical visit scenario (as in hypertext systems). The desirable blind–sighted user collaboration properties are listed in Table 2.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Semantic</td>
<td>Syntactic</td>
</tr>
<tr>
<td></td>
<td>Semantic concepts concern ‘region title’, ‘region content’, ‘number of subregions’ for a particular region. Syntactic concepts concern tasks like ‘reviewing a region’, ‘going one level above in region exploration’, etc.</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Different</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rooms metaphor for non-visual interaction, and desktop windowing metaphor for visual interaction.</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Semi-synchronous</td>
<td></td>
</tr>
<tr>
<td></td>
<td>After a region is selected, the effect is propagated in both interface instances; however, the region selection as well as interaction with region content is facilitated in an asynchronous manner.</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Different</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Rooms non-visual physical structure is radically different from the windowing visual physical structure.</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Different</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The visual and non-visual interface designs expose similarities only at the abstract task design; the syntax, as well as the physical design, diverges significantly.</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>Local</td>
<td>Remote</td>
</tr>
<tr>
<td></td>
<td>Both modes should be supported for users working on the same machine or remotely on independent hosts.</td>
<td></td>
</tr>
</tbody>
</table>
This scenario is of practical added-value: even though existing adaptation-oriented approaches are able to filter information regarding the hierarchical structure of interaction objects in visual applications, they completely fail to reproduce graphical content, which may convey various representations, such as illustrations, pictures, diagrams, charts, or even scanned text. This is because graphics are represented either by composition of lower level graphics primitives (e.g. lines, circles, rectangles, strokes) or they are provided as raster images (i.e. an array of pixels). The problem of access to graphics data became more demanding with the tremendous increase of Web users, since Web pages tend to resemble typical magazine pages, encompassing many graphic illustrations and visual effects that are overlapping with the real text. Work related to Web access for blind users has been carried out, as in pwWebSpeak [21] and V-Lynx [19]. However, none of the systems currently available provides access to graphical data and pictures. In order to tackle the problem of efficient and effective non-visual picture presentation, an approach for the hierarchical annotation of pictures has been designed and demonstrated through a dual application, which has been developed via the HOMER UIMS. Following our annotation model, the image is separated in an arbitrary number of rectangular regions. Each region is characterized by the following attributes: area (upper-left corner and lower-right corner), description (text which describes this image portion), title (a short title for the information that is conveyed by this area) and a list of other image regions which are made accessible when the user explores this region. Some regions are the first to be provided when exploration starts and are characterized by the fact that they do not have a parent region. These are called toplevel regions.

In Fig. 3, the picture annotation model is illustrated. On the basis of this annotation model, the construction of hierarchical picture exploration scenarios for blind users can be carried out by constructing an acyclic visit graph for picture regions. For instance, consider the regions of the picture in Fig. 4 (left part). The visit scenario, regarding the decomposition into regions of this picture, is shown in Fig. 4 (right part). The user should be provided with dialogue facilities to: (i) select a region from those available from the current region to explore; (ii) move to the previously explored region (i.e. ‘back’ operation); and (iii) review the textual information content associated with the current region.

In order to make the life of developers much easier, we put some specific prerequisites on the target interface tool, which are directly relevant to our subject application scenario. As will be discussed later on, those prerequisites are of central importance for dual interface development. First, the tool should enable the user to build interface modules in which both Athena and Rooms/COMONKIT interaction objects are utilized. Second, it is necessary to facilitate the expression of some common syntactic/constructional features, among the visual and non-visual interface instances, into a single abstract form, though still supporting further physical specialization (as needed). Third, the information content as such (i.e. annotations) should be shared among the two interface instances (i.e. replication is not allowed). Finally, changes in the picture exploration dialogue for the sighted user should be allowed, having no implications on the non-visual dialogue implementation (and vice versa), although still preserving the requirement for dialogue synchronization and effect propagation (i.e. when the blind user selects a region to review, the visual interface instance is accordingly updated to display the newly selected region).

Starting from those requirements, the discussion regarding the various features of the
Fig. 3. The hierarchical picture annotation model.

Fig. 4. An image split into regions (left part), and an example visit scenario (right part).
HOMER UIMS will be elaborated. Additionally, some more detailed requirements will be further identified and analysed, while their particular importance in supporting dual interface development will be demonstrated.

4. Dual interface development features

The HOMER UIMS provides the HOMER language for dual interface development. This language is a marriage of interface languages with 4th Generation Languages (4GLs). Interface languages are different from mainstream programming languages like C++ and Java\(^\text{®}\), in the sense that interaction-specific concepts (e.g. interaction object, attribute, method, event, notification, event handler, object hierarchy) are supported as built-in language constructs. 4GLs exist for various programming paradigms, such as procedural, Object Oriented (OO), functional, formal and logic techniques, while they are characterized by the introduction of more declarative algorithmic constructs than those typically supported in the original programming paradigms. The HOMER language is closer to the procedural and OO paradigm, while it introduces various declarative program control constructs such as preconditions, constraints and monitors.

The choice of providing a language-based development method has been dictated by the need to support dual interface development. If the manipulation of both visual and non-visual interaction elements is to be supported, while still being open with respect to integrated physical interaction toolkits, interactive construction techniques must be excluded: on the one hand, a particular interaction technology has to be presupposed in interactive building tools (e.g. OSF/Motif, Windows, X/Athena), while on the other hand, shared specification of common subdialogues is not facilitated in interactive construction, since separate visual and non-visual physical subdialogues need to be explicitly defined. Also, syntax-oriented methods, such as task notations [22] and action grammars [23], do not suffice for developing dual interfaces, since in dual interaction it should be allowed to potentially have two distinct syntactic designs (possibly being very ‘close’, but still different), though ‘fused’ together in a single running interface. Other techniques, such as event-based models [24] and state-based methods [25], have been rejected because of their fundamentally low-level dialogue control approach (more close to programming) and their inability to support abstraction and polymorphism for abstract objects, two critical functional features for dual interface development.

All these previous considerations have led to the design and implementation of the special-purpose HOMER language, specifically suited in supporting dual interface specification. The features of the HOMER language will be discussed in a problem-centred fashion, trying to indicate the types of real interface construction issues particularly addressed. The dual picture exploration application scenario will constitute the basis of presenting and justifying the importance of the HOMER language mechanisms.

4.1. Development of a non-visual toolkit based on the Rooms metaphor

Currently, there is no software library available which provides non-visual interaction elements for building non-visual dialogues. This may seem surprising, but although there
are various screen-readers and custom-made non-visual interactive applications, the analogy of an interface toolkit, like the OSF/Motif or Athena widget set, is clearly missing for non-visual interaction. As a result, developing non-visual interaction applications is not possible, since the necessary software building blocks are completely lacking. However, when initiating the design and implementation process of the non-visual picture exploration dialogue, the need to directly engage pre-existing non-visual interaction elements emerges. This fact has driven a process of designing and implementing a genuine non-visual toolkit (i.e. one that is not influenced by visual concepts), based on a purposefully designed version of the Rooms metaphor, called COMONKIT (briefly introduced in Ref. [6]): this toolkit has also been integrated within the HOMER UIMS.

4.1.1. Rooms metaphor for non-visual interaction

The non-visual toolkit which has been developed and integrated within the HOMER UIMS is based on a non-visual realization of the Rooms metaphor (see Fig. 5), through speech/Braille output and keyboard input. A software implementation of this metaphor has been provided by means of a non-visual interface development library called COMONKIT. The only container object in COMONKIT is the Room object class. A Room object may have an arbitrary number of ‘children’ object instances, including objects of the Room class. There are six groups under which a child object may belong: floor, ceiling, front wall, back wall, left wall and right wall. In the case that an object which belongs to the vertical walls is of Room class, it is presented to the blind user as a ‘door’, which leads to another room. Room objects belonging to the ‘floor’ or ‘ceiling’ groups are made

![Fig. 5. Illustration of the Rooms metaphor for non-visual interaction.](image)
accessible to blind users through the ‘lift’ metaphorical entity, which can go downstairs or
upstairs respectively. Other interaction object classes which have been implemented include:
menu, toggle (represented as on/off switch), button, text reviewer (represented as books), etc.
Particular emphasis was also given to the provision of efficient navigation facilities. For
instance, the blind user is able to move forward/backward in the list of parenthood groups
(i.e. wall groups, floor and ceiling), perform dialogue directly with the next/previous object in
a group, perform dialogue directly with the last or first object in a group, etc.

Two different non-visual realizations of the Rooms metaphor have been assembled: (i) a
non-spatial realization, supporting Braille, speech and non-speech audio output with key-
board input; and (ii) a direct-manipulation spatial realization, combining 3D audio (speech
and non-speech), 3D pointing via a glove and hand gestures, keyword speech recognition
and keyboard input [5]. In both realizations, special sound effects accompany particular
user actions such as selecting doors (e.g. ‘opening door’ sound), selecting the lift (e.g. ‘lift’
sound), pressing a Button or a Switch object, etc.

4.1.2. Navigation dialogue in Rooms-based non-visual interaction

The groups of a Room object are sequentially provided to the blind user during naviga-
tion in the Rooms structure. The order of appearance of the groups can be changed by the
interface developer for each Room (hence, there are $1 \times 2 \times \ldots \times 6$, i.e. 6!, alternatives).
There is a default presentation order which is: FRONT, BACK, LEFT, RIGHT, FLOOR,
CEILING. In Fig. 6, an alternative (to the default) presentation order is shown.

The 3D navigation space is realized as a 3D auditory ring, having the user as the centre.
During the navigation dialogue in the hierarchical Rooms structure, the auditory ring
provides the objects belonging to the particular current Room object; as the user reviews
other Room objects, the auditory ring is updated accordingly. 3D pointing and voice input
is supported for making selections, while presentation techniques to ensure the discrimi-
nation capability of sound effects in the 3D space, even with many items in an auditory
ring, have been designed and implemented (e.g. the virtual auditory ring concept—more
details can be found in Ref. [5]). In Fig. 7, the auditory navigation space and the navigation
dialogue structure are illustrated.

4.2. Integration of physical interaction elements from the target toolkits

In dual interface development, there are two functional requirements concerning the

Fig. 6. An example scenario for the order of presentation of ‘wall’ groups in the non-visual navigation dialogue.
way the underlying toolkits of interaction elements are handled by an interface tool: (a) openness with respect to the toolkits being utilized for visual and/or non-visual interface development, inherently requiring toolkit integration capabilities; and (b) support for combined use of a visual and a non-visual toolkit (i.e. constructing dialogues which consist of both visual and non-visual interaction elements).

In the context of our application scenario, the Athena widget set (for visual windowing interactions) and the COMONKIT toolkit (for non-visual Rooms dialogues) have been imported within the HOMER UIMS. The integration process has been carried out through the toolkit integration mechanism of the HOMER language; interaction elements which are imported via this mechanism maintain the original (i.e. native) ‘look and feel’ of their respective toolkit. The toolkit integration mechanism of the HOMER language realizes a generic toolkit meta-model which is not tied to any specific metaphor of interaction; this

Fig. 7. Non-visual navigation dialogue structure (upper part), and the 3D-auditory ring realization for Rooms navigation dialogue (lower part).
has been practically demonstrated by importing the Athena toolkit, complying to the
desktop windowing metaphor, together with the COMONKIT library, realizing the
Rooms metaphor. The flexibility of the toolkit integration mechanism of the HOMER
UIMS has driven some spin-off developments towards specialized tools, offering merely
toolkit integration services, like the PIM tool (Platform Integration Module—described in
Ref. [26]). According to our generic meta-model, toolkits are considered to support three
fundamental categories of elements: objects, input events and output events. A brief
description of each category is given below:

**Objects** have an arbitrary number of typed attributes (like ‘x’, ‘font’, ‘text’, etc.) and an
arbitrary number of methods (like ‘Selected’, ‘Pressed’, ‘FocusIn’, etc.). Methods typically
characterize what the user is able to do with an object (e.g. selecting an option, pressing the
button). Apart from conventional interaction objects (e.g. ‘Buttons’, ‘Textfields’), objects
can represent graphic interaction elements (e.g. ‘Lines’, ‘Circles’). In this case, the attri-
butes of an object will correspond to the parameters of the particular graphic element (e.g.
‘x’, ‘y’, ‘radius’), while methods will characterize the interactive manipulation facilities
(e.g. ‘Picked’, ‘Moved’, ‘Rotated’, ‘Stretched’).

**Input events** have an arbitrary number of typed parameters carrying event-specific data
and occur asynchronously in the context of objects. Events may concern device-oriented
input (e.g. key presses, mouse moves), notifications for window management events (e.g.
exposure/window destruction in X windows, repainting in WINDOWS), client specific
processing (e.g. user events in both X windows and WINDOWS) that may generate high-
level events (e.g. a gesture recognizer generating gesture events—[27]), etc.

**Output events** have an arbitrary number of ‘out’ parameters (necessary for the toolkit to
perform an output event) and an arbitrary number of ‘in’ parameters (values possibly
returned from the toolkit to the caller, after an output event has been served). Output
events are practically functions of the form $y^n = f(x^m)$, where $x^m$ is the list of out para-
meters (i.e. exported to the toolkit, outgoing values), while $y^n$ is the list of in parameters
(i.e. returned from the toolkit, incoming values). Output events may concern: procedural
graphics primitives and drawing facilities (e.g. ‘POINT_ABS_2’, ‘VIEWPORT_2’,
‘LINE_REL_2’ [28]), toolkit functions for managing interaction objects (e.g. ‘Realize-
Widget’ in Xt, moving/hiding/redisplaying/destroying objects in almost all toolkits),
toolkit resource management (e.g. ‘CreateGC’ in Xt, font management and bitmap manip-
ulation in almost all toolkits), etc.

On the basis of this toolkit meta-model, the HOMER language provides a toolkit inter-
face specification facility, through which all interaction elements from the target toolkit(s),
which need to be imported, have to be defined. Such a (toolkit interface) specification
process establishes a ‘contract’ between the HOMER UIMS and the real toolkit software,
allowing developers to utilize the interaction elements of the underlying toolkits via the
development methods of the HOMER language (i.e. as if those elements have been an
integral part of the HOMER language). At run-time, the HOMER UIMS will dispatch all
references (within dialogue implementation) of imported interaction elements to a corre-
sponding toolkit server, which needs to be developed as part of the integration process.
The role of the toolkit server is to translate all such references and requests to appropriate
commands performed at the toolkit level; hence, it realizes the implementation connection
between the HOMER UIMS and an imported toolkit. When running dual interfaces, two
toolkit servers will need to be activated, providing the connection to the visual and non-visual toolkits respectively.

In Fig. 8, an excerpt from the specification of COMONKIT and Athena imported interaction elements is provided, which are employed for our application scenario. Such a specification serves a twofold purpose: (a) it defines the programming ‘view’ of imported elements for interface developers (e.g. naming conventions, attributes, data types, methods); and (b) it defines the structure of interaction elements as internally communicated at run-time between the HOMER UIMS and toolkit servers (i.e. a conversation protocol).

The prefixes visual or nonvisual indicate whether a particular element defined belongs to the visual or non-visual toolkit respectively. The keywords object, inputevent and outputevent indicate the specification of interaction objects, input events and output events respectively. The keywords method, out: and in: indicate the definition of a method (for objects), list of out parameters (for output events) and list of in parameters (for output events), respectively. Attribute default values are also supported, defined via an assignment on attribute variables within interaction object classes (e.g. soundOnEnter = 'open-door.wav' in Fig. 8).

Fig. 8. Specification of physical interaction elements to be imported from Athena and COMONKIT which are used for the dual picture exploration application.

4.3. Manipulation of virtual interaction objects

The notion of abstraction has gained increasing interest in software engineering as a solution towards recurring development problems. The basic idea has been the establishment of software frameworks clearly separating those implementation layers relevant only to the nature of the problem from the engineering issues which emerge when the problem class is instantiated in practice in various different forms. The same philosophy applied to developing interactive systems means employing abstractions for building dialogues, so that a dialogue structure composed of abstract objects can be retargeted to various alternative physical forms, by means of an automatic process controlled by the developer. Here, we will initially try to draw the complete portrait of what abstraction means in the context of interaction objects, comparing also with previous work and definitions where the concept of abstraction has been explicitly engaged. Then, we will identify the most important functional needs that interface tools should meet, so that manipulation of object abstractions can be maximally supported. Finally, through the realization of some
important development steps of our application example, we will discuss the facilities offered by the HOMER UIMS for efficient and effective management of virtual objects.

4.3.1. Decoupling abstract syntactic behaviour and morphology

Even though the interactive behaviour of interaction objects is characterized by certain properties such as 'look and feel', there are many situations in which differentiated physical realizations reflect similar or even identical forms of behaviour. The level of differentiation varies from simple presentational differences (e.g. consider realization of a 'menu' behaviour for different toolkits such as X/Athena, OSF/Motif, X/InterViews, Windows 95), to radically different morphological structures and interaction policies (see Fig. 9). All four physical interaction objects of Fig. 8 allow some kind of selection from an explicit set of options (i.e. 'menu' behaviour, corresponding to the abstract selector, as illustrated in the diamond in the centre). Consequently, it is possible to decouple the higher-level syntactic behaviour from the specific morpholexical realization (i.e. 'look and feel'), through which the behaviour is physically perceived and understood by the user. Behaviour abstraction techniques, as they have been applied in systems like ADG [10] and Jade [29], mainly relate to toolkit retargetability (i.e. running the same interactive application over different target toolkits). Those schemes of behaviour abstraction are still within the bounds of the desktop metaphor (i.e. they introduce visual attributes, typically met in interaction objects of windowing based environments), while the supplied set of abstract behaviour is fixed (provided as a non-expandable library

![Diagram](image-url)

Fig. 9. Four alternative physical realizations of an abstract selector object: (i) column direct-manipulation menu (restaurant metaphor, upper left); (ii) circular visual structure (clock metaphor, upper right); (iii) auditory 3D pointing (sound-wall metaphor, lower left); (iv) column command-based menu (restaurant metaphor, lower right).
to interface designers). The concept of meta-widgets [15] is closer to the notion of abstract interaction objects; however, the current realization of meta-widgets is restricted to a fixed implementation framework [15], composed of hard-coded programming classes, regarding both the basic meta-widgets categories, as well as their respective physical implementations.

4.3.2. Development requirements for virtual objects in dual interaction

When an interface tool aims to support abstraction of interaction objects, there are certain development requirements which have to be met, so that an abstraction mechanism can be maximally supported in practice [30]:

- Facilities to define new classes of virtual interaction objects (object genesis);
- Methods to define alternative schemes for mapping abstract object classes to physical object classes; e.g. an abstract ‘selector’ may be mapped to a visual ‘column menu’ and a Rooms non-visual ‘list-box’ (open polymorphism);
- Methods to define run-time relationships among an abstract class and its various alternative physical instances; e.g. in Fig. 9, such relationships can be variable dependencies, which are indicated as arrows (physical mapping logic);
- For any abstract object instance, the developer may choose which of the defined mapping schemes will apply, thus having control of the type of physical instantiation to be performed; e.g. may instantiate a virtual valuator, as in Fig. 10, and choose the visual ‘slider’ and the non-visual ‘speech textentry’ mapping/instantiation schemes to be activated (controllable instantiation);
- Ability to associate an abstract object instance with multiple physical object instances at run-time; e.g. when various instantiation schemes are activated for an abstract object instance X, the physical classes associated with each such scheme are instantiated, while those resulting physical object instances are automatically attached to the X abstract object instance (plural instantiation).

The next step is to show how the previous development requirements emerge in practice, when building the dual interactive application for picture exploration. We will discuss those functional requirements together with the corresponding development mechanisms of the HOMER UIMS, by building some important parts of our application subject-case.

4.3.3. Supported specification layers for interaction objects

There are two levels of interaction objects in the HOMER language: (i) the virtual level, which concerns the classes of virtual interaction objects, and (ii) the physical level, which consists of the classes of physical interaction objects. The physical level, apart from providing physical interaction objects that can be directly used for building interactions, plays an additional twofold role. First, it incorporates the description of the interaction objects supported by the toolkits; this is necessary for toolkit integration. Second, it is used for the specification of mapping schemes with which virtual object classes are polymorphically mapped to various physical object classes.

In Fig. 11, the various layers of interaction objects in the HOMER tool are illustrated. At the specification layer, the virtual objects and the physical objects are distinguished. A virtual object class may be mapped to various alternative physical object classes through
Fig. 10. Physical instantiation schemes for polymorphic physical realization of a virtual Valuator object class.

Fig. 11. The various layers of interaction objects in the HOMER UIMS.
the specification of various instantiation schemes. Each virtual object class has two sets of such instantiation schemes, one for the visual and another for the non-visual toolkit. The implementation layer for physical interaction object classes is provided by the toolkit servers, which are built on top of the native toolkit libraries. In Fig. 10, an example of the various intermediate physical instantiation layers is illustrated for an abstract valuator object class. The visual mapping of the valuator class has three instantiation schemes, corresponding to the slider, text-entry and gauge physical classes. The non-visual mapping also has three alternative schemes, mapping to auditory slider, Braille-output text-entry and speech-output text-entry physical classes.

4.3.4. Definition of virtual interaction objects

Virtual interaction objects play a significant role in dual dialogue implementation, since they allow the specification of common syntactic patterns among the visual and non-visual interface instances into a single form. This, on the one hand, reduces the implementation complexity, while, on the other hand, it makes the dialogue implementation less dependent on the target interaction technology utilized (i.e. underlying toolkits). In dual picture exploration, two common syntactic patterns which are identified are: direct execution of an interface supported operation (e.g. ‘back’/exit operations), and selection from an explicit list of options (e.g. ‘select region’). These two dialogue patterns are defined as virtual interaction objects through the genesis mechanism (for creating virtual object classes) of the HOMER language, as shown in Fig. 12. The command dialogue pattern is defined as an abstract Button class, and the selection pattern as a Selector class. The Button virtual object has one attribute, called AllowsDialogue (to enable/disable dialogue with the user), and one abstract method, named Pressed. This is a pure abstraction for all physical objects, independent of physical realization and metaphor, supporting direct actions (like a button-press). Even though its name is directly related to a specific real-world metaphor, in this context it has only a notational value (i.e. it is just a class identifier); a better name may be devised, more indicative of its abstract role.

Similarly, the Selector class encompasses only those attributes which are relieved from physical interaction properties and a single abstract method called Selected. Apart from the AllowsDialogue attribute (which can be seen as a standard property for all virtual classes), it has two additional attributes: NumOfOptions, indicating the total number of items from which the user may select; and UserChoice, which contains the order of the option most recently selected by the user. It might be noticed that the list of options is not stored within the virtual object class; this has been a careful design decision, since options

<table>
<thead>
<tr>
<th>virtual Button</th>
<th>virtual Selector</th>
</tr>
</thead>
</table>
| method Pressed;
  bool AllowsDialogue=true;
  constructor []
  destructor [] |
| method Selected;
  bool AllowsDialogue=true;
  int NumOfOptions=0;
  int UserChoice=-1;
  constructor []
  destructor [] |

Fig. 12. Specification of Button and Selector virtual interaction object classes for use in dual implementation of the picture exploration application.
do not always have a specific information type, but may vary for different physical interaction styles. For instance, options may have textual, iconic, animated, video or audio content. Hence, the storage of options is moved to be part of the physical interaction object classes. For all virtual classes, a constructor/destructor block is supported for code that needs to be executed upon object instantiation/destruction.

4.3.5. Definition of physical instantiation schemes for virtual interaction objects

The instantiation logic of virtual objects is provided via two instantiation relationships: one for the visual toolkit, and another for the non-visual toolkit, which encompass the implementation logic for: (i) mapping an abstract object class to a physical object class; and (ii) defining the run-time relationship between virtual and physical attributes/methods. In Fig. 13, both the visual and non-visual instantiation relationships of the Button abstract class are shown. An instantiation relationship may encompass an arbitrary number of instantiation schemes, which are defined as follows (see also Fig. 13):

1. An arbitrary name is given to each instantiation scheme, which has to be unique across its owner instantiation relationship block. For instance, the name Command_scheme is given to the scheme defined within the visual instantiation relationship, while the name RoomsButton_scheme is similarly given to the only scheme defined as part of the non-visual instantiation.

2. Each scheme maps the virtual object class to a particular physical class. The name of this physical class must be provided, directly following the scheme name. For example, the Command Athena widget class is chosen as the target physical class of the Command_scheme, while the RoomsButton class is similarly defined for the RoomsButton_scheme.

3. In the body of instantiation schemes, the linkage between the virtual and lexical instances is implemented. Normally, constraints and monitors will be employed for bridging together virtual and physical attributes, while method associations define when the virtual methods will be notified due to physical method activation. For example, the physical attribute sensitive (common to Athena widget classes) is constrained by the virtual attribute AllowsDialogue; thus, each time AllowsDialogue is changed, the constraint will be re-evaluated, assigning its value to the sensitive physical attribute. The method association Pressed:Activated defines that each time the physical method Activated is notified for the physical instance (due to interaction), the virtual method Pressed is also to be notified for the virtual instance.

4. The default active instantiation scheme is chosen; in Fig. 13, the expression default

```
visual instantiation Button [  
  Command_scheme : Command [  
    method Pressed : Activated;  
    (me)visual.sensitive =  
      (me).AllowsDialogue;  
  ]  
  default Command_scheme;  
]
```

```
nonvisual instantiation Button [  
  RoomsButton_scheme : RoomsButton [  
    method Pressed : Pressed;  
    (me)nonvisual.accessible =  
      (me).AllowsDialogue;  
  ]  
  default RoomsButton_scheme;  
]
```

Fig. 13. Specification of visual and non-visual instantiation relationships for the Button virtual object class.
Command_scheme defines that Command_scheme is to be activated as the default visual physical instantiation of declared Button virtual object instances (unless another existing scheme is explicitly chosen, as will be explained later).

At run-time, when an instance of a virtual class is declared, the two instantiation relationships of the virtual class are first identified. Then, for each instantiation relationship, the necessary instantiation scheme is activated; this may be the default scheme, or a scheme explicitly chosen by the interface developer (details of declaring virtual objects are provided later). The scheme activation step results in making an instance of the physical class which has been associated with that scheme; this physical instance is attached to the original virtual instance. Finally, the various constructs defined locally within the scheme body are activated, establishing the run-time links among the virtual and the physical object instances. Within an instantiation scheme of a visual (or non-visual) instantiation relationship, the syntactic convention {me}. < attr > provides notational access to the attributes of the virtual instance, while {me}visual. < attr > (or {me}nonvisual. < attr > ) provides notational access to the attributes of the visual (or non-visual) instance respectively.

The virtual object genesis mechanism of the HOMER language supports open polymorphism (i.e. expandable set of alternative instantiation schemes), plural instantiation (i.e. supporting concurrently a visual and a non-visual object instances—dual instantiation), and physical mapping logic definition (i.e. constraints, monitors and method associations in scheme body). We will also show in the next section how controllable instantiation is facilitated, thus fulfilling all the maximal tool requirements for manipulating abstract interaction objects.

4.4. Dialogue agents for dual interaction control

The HOMER language provides an explicit model for ‘packaging’ subdialogues together in dialogue components, called agents. The notion of agents in the HOMER language is related to software architecture agents, like PAC agents [31] and MVC structures [32], while it exhibits no intelligent/autonomous behaviour (as implied in AI interpretations). An agent specification defines an interaction component class; it has to be instantiated somehow (multiple instantiations allowed), to make real interaction component instances. One typical example of interaction components built via agents are dialogue boxes; a dialogue-box agent is defined to encompass various interaction objects (physically comprising the dialogue box), as well as method implementations, event handlers and any other necessary constructs. Making an instance of such an agent will result in instantiating all embedded items (i.e. members of the agent class), thus instantiating interaction objects, registering call-backs and calling any initialization statements.

Agent instances may also be destroyed, in which case all items ‘brought to life’ due to that agent instance will also be cancelled appropriately. Specification of agent hierarchies is allowed, defining control hierarchies, as opposed to part-of hierarchies. In such control hierarchies, an agent instance is automatically destroyed if the parent agent instance is destroyed; hence, agent instances offer a built-in scheme for control-encapsulation and control-dependencies. The HOMER language supports two categories of agent classes,
which differ only in the way agent instances are created or destroyed: (i) agents instantiated by precondition, which may not be directly reused, since within their preconditions they may access either global variables or variables belonging to hierarchically higher agents (however, simple preprocessor facilities may be applied for creating class patterns, i.e. templates, and for customizing to specific declarations when required); (ii) agents instantiated by call, which are explicitly instantiated through an instantiation statement and can have an arbitrary number of parameters (programming variables, virtual/lexical object instances, agent instances). Parameterized agent classes are potentially more reusable than precondition-based agent classes, and may be employed in constructing generic interface components, such as dialogue boxes or domain specific subdialogues.

In this section, we will elaborate on the picture exploration application, by providing a concrete physical design scenario for the visual and non-visual interaction. We will provide the key aspects of the dialogue control implementation via agents, trying to also demonstrate the appropriateness of our agent-based specification approach for dual interaction control.

4.4.1. The visual and non-visual physical designs for dual picture exploration

The visual interface screen snapshot for the picture exploration application is shown in Fig. 14. The picture is displayed on top, while an outline of the currently visited region is illustrated with a solid rectangular outline. The description of the current region is given in a text-display widget (lower left), while the list of titles for available regions from the current region is provided within a list widget (lower right). The design of the non-visual interface is illustrated in Fig. 15, and complies with the Rooms metaphor. The dialogue for picture exploration fits within any ‘wall’ group, so that multiple pictures may be reviewed concurrently (in various wall groups); for clarity, only the ‘front wall’ group is shown in Fig. 15. The picture exploration dialogue is realized via the following objects: (i) a label object, containing the short title of the current region; (ii) a text reviewer object, containing the description (i.e. text content) of the current region; (iii) a menu, providing the short titles of the regions available in the exploration scenario from the current region; and (iv) a command button, to move directly to the previously visited region. At the ‘floor’ group, two standard objects are instantiated: the ‘Exit’ command object and a text reviewer object including general information about the application and the various available pictures which can be explored.

4.4.2. Constructing the dual interface

The designed scenarios will have to be translated to an implementation form through the HOMER language. First, the two interface instances are translated into a single abstract object hierarchy, revealing the ability to employ directly virtual interaction objects for object implementation. In Fig. 16, the abstract object hierarchy, and its respective visual and non-visual specializations, according to the designed artefacts of Figs 14 and 15, are illustrated. For the identification of the abstract object hierarchy, the developers may employ various techniques, such as interaction tasks [28] and hierarchical task analysis [33]. However, such techniques require their explicit engagement as part of the interface design process; hence, it is not possible to apply those techniques after a design process has been concluded, possibly following a different design methodology. Alternatively to
such approaches, designers may employ a re-engineering approach for identifying abstractions, relying on the role-based model [30]. The role-based design technique drives the identification of abstract object hierarchies from physical design scenarios, and promotes the composition of alternative physical designs, starting from the constructed abstract object hierarchy (i.e. a re-engineering philosophy).

The first step is the specification of the interaction objects within an agent class. The abstract hierarchy of Fig. 16 is mapped to a list of virtual object declarations, as shown in Fig. 17. Additionally, some visual interaction objects are declared, providing some spatial layout facilities (i.e. containers). When declaring virtual object instances, the developer may explicitly define: (a) the hierarchical parent object, for any of the two physical
instances created when a virtual object is created (visual parent = and nonvisual parent =); and (b) the visual and/or non-visual instantiation schemes, which provide control of what type of physical object instances will be created due to a virtual object instance (controllable instantiation — visual scheme = and nonvisual scheme = ). For physical objects, like the visual box and image, the parent object does not need the qualifiers visual or nonvisual, thus the expression parent = is applicable.

The next step is the injection of behaviour (i.e. call-back logic) on virtual interaction objects. In Fig. 18, the implementation of a virtual method is shown. The code supplied to such a method will be executed, if either the corresponding physical methods of the associated visual or non-visual instances are notified during interaction. More specifically,
if the blind user (or sighted user) presses the Rooms ‘back’ button (or Athena ‘back’ Command), the virtual `Pressed` method will be notified. Hence, both situations have been captured into a single implementation, being that for the virtual method. What remains to be done is to also accomplish shared specification of the interface effect that the ‘back’ operation will have, in both concurrent interface instances. This is accomplished via the implementation of the `ShowRegion` function. This function updates accordingly the attributes of virtual objects, thus ensuring that there will be an automatic dual propagation effect on the visual and non-visual instances. There are some other dialogue control details, like the visual feedback with the region highlighter and audio effects, which are omitted for clarity; these are also included within the `ShowRegion` function.

4.4.3. Updating the physical interface instances—the ‘visual case’

Even though shared specification of subdialogues may be enabled for a large part of the designed dialogue artefacts, there are certain situations where some interactive features

```plaintext
bool start=false;
agent DualInterface create if (start=true) {
  virtual Container app;
  :visual scheme=ApplicationWindow :nonvisual scheme=Room;
  visual Box box :parent={app} :visual;
  virtual Selector titles :
  :visual scheme=List
    #define PARENT_OBS
    :visual parent={Box} :nonvisual parent={app} :nonvisual
  virtual Text content PARENT_OBS;
  virtual Command back PARENT_OBS;
  virtual Command exit PARENT_OBS;
  visual Simple image :parent={box};
  constructor [...] destructor [...] };

Fig. 17. A precondition-based agent class definition, encompassing declarations of virtual object instances, for dual implementation of the picture exploration interface structure.

Fig. 18. (a) Shared data structures for holding region information; (b) a dual presentation function, updating attributes of virtual objects and their respective physical instances (for a `Selector titles` instance, the physical instances are used to assign a list of region titles); (c) shared behaviour specification for the ‘back’ operation, working for both visual and non-visual instances.
```
need to be added exclusively, either to the visual or the non-visual interface instances. This
implies the provision of dialogue control logic directly at the physical level of interaction.
In our application, one such example concerns the provision of mouse-based direct region
selection facilities to the sighted user, from the displayed image. The development require-
ment that has to be preserved is to keep the non-visual interaction control implementation
completely untouched, while still ensuring dual effect propagation (i.e. visual region selec-
tion will have a dual effect). The solution is shown in Fig. 19; a visual event handler is added
in the main agent, for mouse-based region selection, while the
ShowRegion function is
called, which will propagate the region selection effect into both interface instances. The
implementation is ‘clean’, while the extra visual interaction control is well encapsulated
within the DualInterface agent class. Another typical case in which interface-related code
has to differentiate, concerns setting attributes of physical interaction objects via the virtual
instance variables. This is also indicated in Fig. 18; the convention \{<\ obj >\}.visual,
where <\ obj > is a virtual object instance, provides access to the visual physical instance, while
the expression \{<\ obj >\}.nonvisual represents the non-visual instance.

The same syntactic convention applies to methods as well. It is possible to implement a
physical method directly, thus adding non-dual behaviour to virtual objects; if this is used
in accordance with virtual methods, it adds great flexibility in the implementation of dual
interaction. For instance, assume that an extra visual effect is to be added on the visual
interface instance for the ‘back’ operation. Then, it suffices to add the code implementing
this visual effect as a visual method implementation (see Fig. 19, last line), without
affecting the rest of the dual interface behaviour.

4.5. Application interfacing

The application interface model combines (i) the shared space model, where arbitrary
data structures can be exported, and (ii) the model of typed communication channels, where an arbitrary number of direct strongly typed links can be created for information exchange. Even though these two models are expressively equivalent, the application convenience varies depending on the conversation protocol which has to be mapped in the model. For instance, the object-oriented approach, which is based on the information structures manipulated by the functional core, is more naturally realized through the shared space approach, while the (older) procedural approach of treating the application as a semantic server is more easily specified by means of direct messages which carry commands and parameters. The coordination scheme is asynchronous, but synchronous behaviour can be modelled; for instance, the dialogue control can wait for a notification that an action has been completed by the functional core and vice versa. Conceptual abstraction or adaptation of the internal structures and services is facilitated. In Fig. 20, specification of a simple application interface structure is shown (left column), including statements (right column) for sending messages (lines 1, 2), creating (line 5) and reading (line 6) shared objects. Even though this specific approach for application interfacing in the HOMER language is not directly related to the notion of dual interaction, it has been adopted as a powerful method for handling the communication needs with the functional core.

4.6. Run-time architecture

The HOMER run-time architecture is based on the Dual run-time model (see Fig. 21) which forms an extension of the Arch model [34] for handling a dual lexical layer. The Arch model introduces the concept of a generalized ‘Presentation Component’ which stands a level above platforms; however, the concept of ‘Presentation Objects’ as the only entities communicated between the dialogue control and the ‘Presentation Component’ is not considered completely adequate. For instance, direct handling of device input events or even specific output functions, which cannot be easily modelled in an object-oriented fashion (e.g. the user selects a region of the screen and copies it as a bitmap to another location), is not covered by the Arch model. Our model in the HOMER UIMS, which is reflected in the toolkit integration specification mechanism of the HOMER language, supports explicitly input and output events.

The Application Server, the Dialogue Manager, the Application Interface (API) and the toolkit servers are implemented as separate processes. The toolkit server interfaces (VI–TSI and NV–TSI) are not implemented as separate processes (as is the case with the API which handles the shared space and message channels). Instead, the communication
interface between the Dialogue Manager and the toolkit servers is realized on the basis of a general and efficient protocol (the generic toolkit interfacing protocol—GTIP, defined in Ref. [26]) which explicitly introduces entities such as interaction objects and input/output events. The protocol implementation is provided to toolkit server developers by means of a class library, which hides lower level communication aspects.

4.7. Remote and local collaboration modes supported

Dual interfaces developed through the HOMER system may run in two different modes: local collaboration and remote collaboration. These two modes are illustrated in Fig. 22, where the devices supported by the non-spatial realization of the Rooms metaphor are shown for non-visual interaction. In the case of the local collaboration mode, because the keyboard is shared, a turn taking dialogue has been developed. During a dual interaction session, when the blind user is in charge of interaction, the mouse pointer is ‘locked’ in a terminal window reserved by COMONKIT (i.e. this terminal window is always the keyboard listener window), while the sighted user is not allowed to change this status.

The blind user is provided with a special purpose navigation command for passing interaction control to the sighted user; this command actually ‘unlocks’ the mouse pointer from the non-visual input window, thus enabling the sighted user to continue interaction in the overall windowing environment. In a similar manner, the sighted user may pass control

Fig. 21. The dual run-time architectural model of the HOMER UIMS.
to the blind user by clicking the mouse over the non-visual input window; the mouse pointer will be automatically locked and the blind user will be informed to continue interaction.

It should be noted that this ‘locking’ dialogue technique applies only in the context of local collaboration mode, and is used to avoid conflicts in input–device use. Such conflicts may arise if the blind or the sighted user accidentally moves the mouse pointer outside the non-visual input terminal window, while the blind user is ‘in charge’ of performing interaction. This ‘locking’ concept is different from traditional CSCW mechanisms, such as ‘locking’ and ‘floor control’, which are applied on remote collaboration to ensure consistency on shared items manipulated by multiple users concurrently.

5. Evaluation of the picture exploration application

We have evaluated the picture exploration application with five blind users, assessing primarily their subjective opinion regarding the interactive facilities supplied. The essence of subjective usability evaluation is the collection, analysis and measurement of the subjective opinion of users when using interactive software. Therefore, it does not rely upon an expert’s opinion, or that of any other intermediary evaluation actor. There are currently several techniques available for subjective evaluation, including structured or unstructured interviews and the use of diary studies, as well as talk-about methods, while more recently, questionnaire techniques have been successfully introduced and widely applied.

Typical examples of such questionnaires are the QUIS questionnaire [35], the SUMI questionnaire [36] and the IBM Computer Usability Satisfaction Questionnaires [37]. The latter instrument, which was the one selected for this evaluation, measures the user’s subjective opinion in a scenario-based situation of use. Two types of questionnaires are typically employed: the first, named After Scenario Questionnaire (ASQ), is filled in by users each time a particular scenario is completed (so it may be used several times during
an interactive session); the second, named Computer System Usability Questionnaire (CSUQ), is filled in at the end of the evaluation (one questionnaire per user).

The selection of the IBM Usability Satisfaction Questionnaires was based on the following two reasons: first, they are publicly available, while the rest require the acquisition of a license from their vendors; and second, and most importantly, because they have shown to be extremely reliable [37]. The result of the subjective evaluation with the IBM Computer Usability Satisfaction Questionnaires is a set of psychometric measurements which can be summarized as follows:

- ASQ score for a participant’s satisfaction with the system for a given scenario;
- OVERALL metric provides an indication of the overall satisfaction score;
- SYSUSE metric provides an indication of the system’s usefulness;
- INFOQUAL metric is the score for information quality;
- INTERQUAL metric is the score for interface quality.

We have organized the evaluation process on the basis of three interaction scenarios, in which users have been asked to accomplish specific tasks. All five users have been supplied with the same interactive picture accessed via the picture exploration application—the organizational chart of an enterprise. Each user had to perform the following three scenarios:

- Identify the summary of the enterprise profile;
- Identify the corporate members at a particular management level, as well as the president of the enterprise;
- Identify the total number of management layers.

Following the evaluation during the interactive sessions, all users responded to a small questionnaire intended to extract some information regarding their background and experience, as well as familiarity with the type of interactions which can be built through the HOMER UIMS. Such information has been quite useful in interpreting some of the results, and has been collected on the basis of the following items:

- Familiarity with the Rooms/COMONKIT non-visual interaction metaphor;
- Background and computer experience;
- Prior use of any particular software system for blind user access, such as screen readers;
- Whether they had access to any documentation or instruction material regarding the picture exploration application;
- Whether they have been offered any guidance during the execution of the scenarios.

On the basis of the above plan, the evaluation process has been carried out with five blind users. Prior to the evaluation, all users have been provided with a summary of those scenarios, while all users involved had been familiar with computers (have used IBM Screen Reader® for text-mode access on PCs). Also, the Rooms metaphor was introduced briefly to all users. One of the users had considerable experience in using a screen reader for Windows® supporting spatial reproduction of the visual graphical environment. After the evaluation process, the data collected were analysed to calculate the metrics [37] supported by the adopted usability evaluation method; the relevant results are summarized...
From those results, it follows that the subjective opinion of users regarding the picture exploration application built through the HOMER UIMS is quite good, while it was observed that three users gave more positive results than the other two users. The profiles of the latter two users provided some explanation regarding their respective responses: (a) one using a Windows screen reader for about a year, having some problem in getting familiar with non-spatial, non-windowing navigation, complaining that more training was required; and (b) one having less computer experience, found it difficult to assimilate the notion of ‘interaction objects’ so quickly.

In conclusion, users appreciated the quality of the interactive features, as well as the specific capabilities offered by the picture exploration application, while most of the concerns and suggestions for improvement were related to the introduction of extra functionality such as: searching phrases, ability to save parts of text in a file, etc. Finally, one of the users that gave the most positive responses had been just recently engaged (four days prior to the evaluation process) in a training seminar for using a commercial Windows screen reader; he expressed his concern for having to learn a visually-oriented metaphor, while he found the picture exploration application very useful, as well as generally easy to learn and use.

6. Discussion and conclusions

Existing methods for providing access to graphical user interfaces by blind users suffer from many theoretical and practical problems. Current systems filter the visual windowing environment and reproduce running interactive applications in a non-visual form. Such a reproduction of visual interaction is based on a hard-coded, non-visual interaction style.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>User</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>2.5</td>
<td>3</td>
<td>2.8</td>
<td>3.5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Scenario 2</td>
<td>2.2</td>
<td>3</td>
<td>3</td>
<td>3.5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Scenario 3</td>
<td>2</td>
<td>3.5</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Table 3
ASQ scores from the subjective usability evaluation of the picture exploration application
that these systems adopt; when the non-visual interaction style for various applications needs to become different, some fine-tuning/configuration facilities are provided for affecting mainly simple presentation parameters. It is evident that the traditional iterative design and implementation cycles for non-visual interaction are not supported in automatic reproduction, since the starting point is the original visual design. Moreover, the configuration facilities are very limited if someone wants to apply specific design features within the non-visual dialogue resulting from the automatic reproduction of a particular visual interactive application (e.g. adding two confirmation levels for critical operations in a process control application).

It is argued that the main cause of the present situation has been the lack of tools for supporting development of user interfaces meeting the specific blind user needs, and supporting the complete cycle of non-visual interface development. In this context, the concept of dual user interfaces has been proposed and defined; it is considered as an appropriate basis for ‘integrating’ blind and sighted users in the same working environment. By analysing the development needs of dual interfaces, the following implementation mechanisms have been considered of key importance: (i) abstraction of interaction objects; (ii) concurrent management of at least two toolkits; (iii) meta-polymorphic capability for abstract objects (i.e. can be mapped to more than one toolkit, can be mapped to more than one object class within a specific toolkit); (iv) unified abstract object hierarchies supporting different physical hierarchies (i.e. physical parent differentiation); (v) ability to integrate different toolkits; (vi) object-based and event-based model support for dialogue implementation; and (vii) declarative asynchronous control models (e.g. preconditions, monitors, constraints), as opposed to syntax-oriented control models. It has been considered that the above advanced development requirements can be more efficiently and effectively managed in the context of a 4GL-based interface development. The lack of an interface specification language providing these specialized features has driven the design of the HOMER language, and the development of the HOMER UIMS. The HOMER language includes mainly special purpose specification mechanisms reflecting the above development requirements, while it borrows some features from other languages (e.g. application interfacing method based on channels and shared space theory for reactive systems, programming kernel based on a C subset, event model based on enhanced ERL, agent parameterization taken from OOP classes).

The lack of a non-visual toolkit to support non-visual interface development posed the necessity of explicitly developing one (called COMONKIT), which has been integrated within the HOMER UIMS. The COMONKIT library has been developed on the basis of a purposefully designed version of the Rooms metaphor, with particular emphasis on providing efficient navigation facilities.

The HOMER UIMS has been utilized for building various dual interactive applications in the context of the GUIB project [7], such as a payroll management system, a personal organizer and an electronic book with extensive graphical illustrations and descriptions. The most encouraging and positive reaction has been that of non-visual interface designers engaged in the application tests of the GUIB project, for two reasons: (a) their role in the development process was fully substantiated via the HOMER UIMS, and (b) they found the necessary construction facilities in order to start experimentation and testing with new interaction techniques, paradigms and design patterns, independently of the visual physical design.
Acknowledgements

This work has been partially funded by the TIDE Programme of the European Commission (DG XIII), under the projects GUIB (TP 103) and GUIB-II (TP 215). The partners of the GUIB consortium are: IROE-CNR, Italy; Institute of Computer Science—FORTH, Greece; Vrije Universiteit Brussel, Belgium; Department of Computer Science—FUB, Germany; Institute of Telecommunications—TUB, Germany; IFI, University of Stuttgart, Germany; VTT, Finland; RNIB, UK; F.H. Papenmeier, Germany.

References


