Universal Access in the Information Society manuscript No.
(will be inserted by the editor)

Equivalent representations of multi-modal user interfaces

Runtime Reification of Abstract User Interface Descriptions

Kris Van Hees · **Jan Engelen**

Received: date / Accepted: date

Abstract While providing non-visual access to graphical user interfaces has been a topic of research for over 20 years, blind users still face many obstacles when using computer systems. Furthermore, daily life has become more and more infused with devices that feature some kind of visual interface. Existing solutions for providing multi-modal user interfaces that ensure acecssibility are largely based on either graphical toolkit hooks, queries to the application and environment, scripting, model-driven user interface development or runtime adaptation. Parallel user interface rendering (PUIR) is a novel approach based on past and current research into accessibility, promoting the use of abstract user interface descriptions. Based on a single consistent conceptual model, PUIR provides a mechanism to render a user interface simultaneously in multiple modalities. Each representation of the UI provides equivalent semantics to ensure that collaboration between users of different modalities is facilitated. The possible application of this novel technique goes well beyond the realm of accessibility, as multi-modal interfaces become more popular and even needed. The design presented here has been implemented as a prototype for testing and further research in this fascinating area of HCI.

Keywords Accessibility, UIDL, Universal Access, multi-modal interfaces, HCI

1 Introduction

Over the past few years, our world has become more and more infused with devices that feature a graphical user interface, ranging from home appliances with LCD displays to mobile phones with touch screens and voice control. The emergence of GUIs posed a complication for blind users, due to the implied visual interaction model. Despite the fact that GUIs have been in existence for about 25 years, the blind still encounter significant obstacles when faced with this type of user interfaces. On UNIX-based systems, where mixing graphical visualisation toolkits is common, providing non-visual access is even more problematic. The popularity of this family of systems keeps growing, while advances in accessibility technology in support of blind users remain quite limited. Common existing solutions depend on toolkit extensions, scripting, and complex heuristics to obtain sufficient information in order to build an off-screen model (OSM) as a basis for non-visual rendering [68,62,35]. Other approaches use model-driven UI composition or runtime adaptation [21,61].

Alternatives to the graphical user interface are not an exclusive need of the blind. Daily life offers various situations where presenting the UI in a different modality could be a benefit. Sometimes the problem at hand is as simple as sunlight glare on the screen of an Automated Teller machine (ATM); other times one might be faced with the dangers of operating a cellular phone while operating a vehicle [38]. In addition, consider the use of computer displays in operating rooms where a surgeon certainly would prefer not turning away from their patient in order to access some information on the screen.

All these situations are very similar to the needs of a blind individual trying to access a computer system. Alan Newell introduced the very important notion that by embracing the needs of extra-ordinary people, one does not limit the applicability of the work [47]. Rather, this provides opportunities for advances that benefit the general public. The presence of higher level technologies in environments where conditions may be variable contributes to the applicability of

Katholieke Universiteit Leuven
Department of Electrical Engineering
ESAT - SCD - DocArch
Kasteelpark Arenberg 10
B-3001 Heverlee, Belgium
E-mail: kris@alchar.org, jan@docarch.be

multimodal interfaces. Consider automobile entertainment systems, where the driver certainly should prefer not having to operate on-screen controls or avert their eyes from the roadway in order to read information on a screen. Being able to present the information through synthetic speech, and providing multiple data entry methods allows for the possibility of safe operation of in-car electronics [42].

Past and current research indicates that abstracting the user interface offers a high degree of flexibility in rendering for a multitude of output modalities. Leveraging the well established paradigm of separation between presentation and application logic [49], advances in multimodal UI development, and existing research on abstract user interface descriptions, this work presents a novel approach to providing equivalent representations of multimodal user interfaces, using a parallel rendering technique to support simultaneous representation and operation of the UI across different modalities. The user interface is described in a UIDL document, and reified at runtime along multiple parallel paths.

The remainder of this article first introduces some important terminology as it relates to this work in section 2, followed by a presentation of a reference framework that can be applied to past and current approaches to providing multimodal user interfaces in section 3. The state of the art is discussed is section 4, where further requirements for the approach presented in this article are formulated. The design of the Parallel User Interface Rendering approach is presented in section 5. Conclusions and future work are discussed in section 6.

2 Terminology

Many concepts and terms used in the field of HCI and related areas lack clear and consistent definitions for which consensus has been reached. This section provides definitions as they are applicable to the presented work.

2.1 Blindness and visual impairment

Various terms relating to blindness and visual impairment are used in research literature. By its very nature, visual impairment covers a very broad area ranging from no light perception at all to blurred vision, and every gradation in between. Light perception relates to the ability to, e.g., determine through vision whether one is in a dark or bright location. In addition, the field of view may be restricted or it could include so-called blind spots.

The term "legally blind" is used to indicate that someone meets a specific set of criteria based on either low acuity or a restricted field of vision. The criteria differ from country to country; in the United States of America, legal blindness is defined as having a visual acuity of 20/200 or less in the better eye, with the use of a correcting lens, or a field of vision where the widest diameter subtends an angular distance of 20 degrees or less in the better eye [15].

The term "low vision" is used to "describe individuals who have a serious visual impairment, but nevertheless still have some useful vision" [26].

The term "blind" is often used in a restrictive sense to indicate that someone's vision is limited to light perception or less. Individuals who are deemed "blind" do not have any usable vision.

The term "visually impaired" is used for any individual who is deemed legally blind [40].

2.2 Usability

Literature has defined "usability" in many different ways, often due to differences in view point and context. Various international standards have also provided definitions that are not quite consistent with one another, e.g., IEEE Std.610.12 [27], ISO/IEC 9126 [28], and ISO 9241-11 [29]. In the context of multimodal user interfaces, the latter definition is most on target. It states [29, p. 2]:

> "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

It specifically identifies three areas of concerns that are to be evaluated towards qualifying usability: effectiveness, efficiency and satisfaction. When considering multiple target user groups (e.g., groups with differing abilities and needs), it is important to be able to measure usability as a success criterion. Achieving equivalent levels of usability (measured by a common standard) is a long-term goal for the approach presented in this work.

2.3 Accessibility

Bergman and Johnson define "accessibility" as follows [3]:

> "Providing accessibility means removing barriers that prevent people with disabilities from participating in substantial life activities, including the use of services, products, and information."

This definition is unfortunately not specific enough for the work presented here. Mynatt recognised that an important aspect of accessibility often gets overlooked or is taken for granted [44]: "An implicit requirement [is to] facilitate collaboration among sighted and blind colleagues. [. . .] Therefore it is imperative that [they] be able to communicate about their use of application interfaces." The barrier to collaboration between sighted and blind users is often overlooked

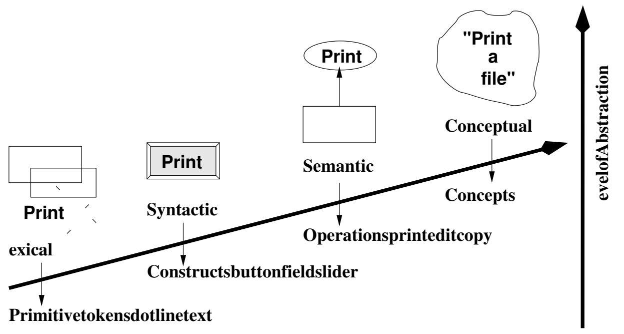


Fig. 1 Four distinct layers of user interface design

when considering the accessibility and usability of computer systems, which is a sad irony in view of the current proliferation of distributed work environments where workers are no longer in close proximity to one another. In addition, the aforementioned definition of accessibility does not quite make the requirement for "usability" explicit, even though it is of great significance [67].

Specific to the context of computer systems, a more refined definition of "accessibility" can therefore be formulated:

> A computer systems is fully accessible when (a) any user can access and use all functionality independently[1], (b) when that user can engage in meaningful collaboration about the system with peers, regardless of individual needs, and (c) when all users are provided with an equivalent level of usability.

2.4 Multimodality

Within the context of this work, multimodality is more a characteristic of the actual system than an aspect of user interaction. Nigay and Coutaz provide a definition for multimodality from a system centric point of view [48]:

> Multimodality is the capacity of a system to communicate with a user along different types of communication channels and to extract and convey meaning automatically.

3 Reference framework

While UI design is often thought of as a self-contained and straightforward process, four distinct layers of design can be identified (see Fig. 1) [30,20][2], and in view of Gaver's work concerning multiple layers of metaphor and their mappings, they can be grouped together in function of the layer of metaphor (or model) they operate on:

- Conceptual metaphor layer
  - *Conceptual*
    This layer of design describes the elements from the physical world metaphor that are relevant to the UI. It also describes the manipulations that each element supports.
  - *Semantic*
    This layer of design describes the functionality of the system, in an abstract way, independent from any specifics concerning user interaction. It defines the operations that can be performed in the system, and provides meaning to syntactic constructs in a specific UI context.
- Perceptual metaphor layer
  - *Syntactic*
    This layer of design describes the operations necessary to perform the functions described in the conceptual layer. This description is modality specific, using the fundamental primitive UI elements to construct a higher order element that either enables some functionality or encapsulates some information. Examples are: buttons, valuators, text input fields, etc. These elements are presented to the user as entry points of interaction to trigger some aspect of the system's functionality as defined at the conceptual layer.
  - *Lexical*
    This layer of design maps low-level input modality operations onto higher level fundamental operations of UI elements. At this level, the UI is expressed as a collection of primitive elements, such as dots, lines, shapes, images and text.

The distinction between the different layers and their grouping is important in consideration of accessibility, because solutions will generally encompass a specific layer and all those below it (usually in the order of the list above). Adaptations at the lexical level may involve the use of haptic input devices or a Braille keyboard, whereas assistive technology solutions usually operate at the syntactic level and affect the perceptual layer as a whole. Common examples are various screen readers. Approaches to accessibility at the semantic or conceptual level are less common, because they usually require an adaptation at the level of application or system functionality.

[1] Either through direct manipulation ("direct access") or indirectly ("assisted access") by means of some form of assistive technology solution.

[2] Edwards, Mynatt, and Stockton list only three layers in [20], but they limited themselves to a description of the layers of modelling, where the conceptual layer forms the basis for those three layers of modelling.

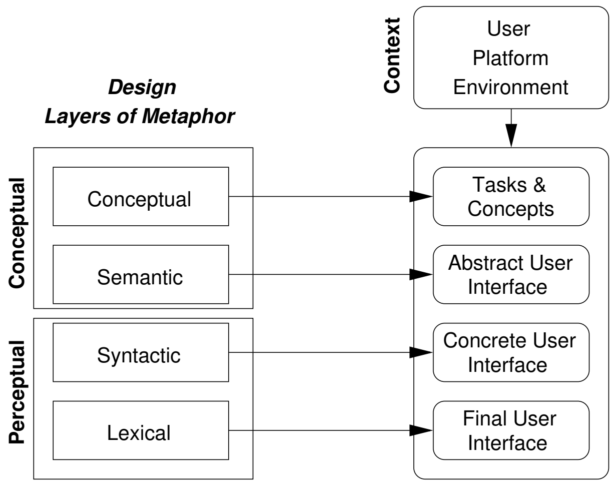


Fig. 2 Design layers in the Unifying Reference Framework

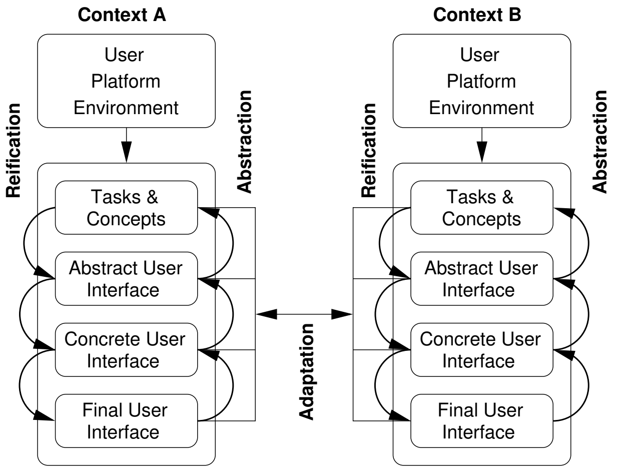


Fig. 3 Unifying Reference Framework

3.1 The Unifying Reference Framework

Calvary, et al. developed a Unifying Reference Framework[3] (URF) [11,13,12] for multi-target user interfaces, specifically intended to support the development of context-aware UIs[4]. The context of use in this framework comprises three components: a target user population, a hardware/software platform, and a physical environment. Each aspect of the context of use may influence the UI development life cycle at any of four distinct levels. The levels of abstraction recognised in the Unifying Reference Framework correspond to the four layers of UI design (see Fig. 2):

– *Tasks & Concepts* (T&C): User interface specification in terms of tasks to be carried out by the user and well-known underlying concepts (objects manipulated during the completion of tasks).
– *Abstract User Interface* (AUI): Canonical expression of the interactions described at the the T&C level. The interactions can be grouped to reflect logical relations most often seen in multi-step tasks and sequences of tasks.
– *Concrete User Interface* (CUI): Specification of the UI in terms of a specific “Look & Feel”, but independent from any specific platform. The CUI effectively defines all user interaction elements in the UI, relations between the elements, and layout.
– *Final User Interface* (FUI): The final representation of the UI within the context of a specific platform. This is the actual implementation of the UI. It may be specified as source code, compiled as a pre-computed UI, or it may be instantiated at runtime.

The development of a UI (as modelled in the Unifying Reference Framework) can be accomplished by means of transformations between the aforementioned levels. Both top–down and bottom–up transformations are possible, depending on the initial design.

– *Reification* (top–down): A derivation process whereby an abstract specification is made more concrete.
– *Abstraction* (bottom–up): A reverse engineering process whereby an abstract specification is inferred from a more concrete one.

By means of these two operations, the framework is able to model a large variety of UI development processes. E.g., a designer might prototype a UI at the concrete level using a design tool. In this case, reification will yield the final UI, whereas abstraction provides for the specification of the user tasks and underlying concepts.

When multimodal user interfaces are considered, the development of the UI spans multiple contexts of use. The Unifying Reference Framework supports this with the addition of a third operation (see Fig. 3):

– *Adaptation* (cross context): A transformation process in which a UI specification at a given level for a specific context of use is translated to a UI specification (possibly at a different level of abstraction[5]) for a different context.

3.2 The CARE properties

In the context of multimodal user interfaces, possible relations between modalities may exist. Coutaz, et al. [16] define

[3] Also known as the CAMELEON Reference Framework.
[4] The Unifying Reference Framework comprises more elements than are presented here. The discussion of the state of the art does not require all elements of the framework, and the scope has therefore been limited to what is sufficient to describe, understand, and compare the various approaches.

[5] The initial version of the Unifying Reference Framework [11] defined the adaptation operation as a transformation between representations at the same level of abstraction. Revisions made in support of plasticity of user interfaces (being able to adapt to context changes without affecting usability) introduced adaptation as a cross level operation.

a set of four properties to characterise these relations: Complementarity, Assignment, Redundancy, and Equivalence.

The formal definition of these properties is based on the following important concepts:

- *State*: A set of measurable properties that characterise a situation.
- *Interaction trajectory*: A sequence of successive states.
- *Agent*: An entity that can initiate the execution of an interaction trajectory.
- *Goal*: A state that an agent intends to reach.
- *Modality*: An interaction method that an agent can use to reach a goal.
- *Temporal relationship*: A characterisation for the use of a set of modalities over time.

3.2.1 Equivalence

The equivalence property expresses that interaction trajectory $s \rightarrow s'$ can be accomplished using any of the modalities in set M. It therefore characterises a choice of modality. It is important to note that no temporal constraint is enforced, i.e., different modalities may have different temporal requirements for completing the interaction trajectory.

3.2.2 Assignment

Contrary to the equivalence property, assignment characterises the absence of choice. A given modality m is said to be assigned to the interaction trajectory $s \rightarrow s'$ if no other modality is used for that trajectory, either because it is the only possible modality (StrictAssignment), or because the agent will always select the same modality m for the trajectory (AgentAssignment).

3.2.3 Redundancy

The redundancy property characterises the ability to satisfy the interaction trajectory $s \rightarrow s'$ with any of the modalities in set M within temporal window tw. Redundancy comprises both sequential and parallel temporal relations.

3.2.4 Complementarity

The complementarity property states that the modalities in set M must be used together in order to satisfy the interaction trajectory $s \rightarrow s'$, i.e., none of them can individually reach the goal.

3.3 Non-visual access to GUIs

Mynatt, Weber, and Gunzenhäuser [46,23] formulate five important HCI concerns that need to be addressed in order for an approach towards providing non-visual access to a GUI to be deemed a viable solution. These concerns translate to necessary requirements under the accessibility definition in section 2.3, yet it is recognised that they do not constitute a sufficient set of requirements.

- Coherence between visual and non-visual interfaces
- Exploration in a non-visual interface
- Conveying graphical information in a non-visual interface
- Interaction in a non-visual interface
- Ease of learning

Weber further specified coherence in two different forms [67]:

- *Static coherence*: A mapping between all visual and non-visual objects, which lets users identify an object in each modality.
- *Dynamic coherence*: A mapping that defines for each step in interaction within the visual modality one or several corresponding steps within the non-visual modality. This form of coherence satisfies the “Equivalence” CARE property presented in section 3.2.

The “ease of learning” concern will not be addressed in this article.

4 State of the art

This section presents past and current approaches to multimodal user interfaces in the field of HCI. While accessibility was not the primary concern for several of these projects, the proposed tools and techniques most certainly can be applied to this problem. This section highlights some of the influential approaches from the field of Human-Computer Interaction. First, an overview of approaches is presented. Then, two representative projects are discussed in greater detail: the “Fruit” project in section 4.1, and the “HOMER UIMS” in section 4.2.

Accessibility of GUIs for blind users has been a topic of research for many years. Mynatt and Weber discussed two early approaches [46], introducing four core design issues that are common to non-visual access to GUIs. Expanding on this work, Gunzenhäuser and Weber phrased a fifth issue, along with providing a more general description of common approaches towards GUI accessibility [23]. Weber and Mager provide further details on the various existing techniques for providing a non-visual interface for X11 by means of toolkit hooks, queries to the application, and scripting [68].

Blattner et al. introduce the concept of MetaWidgets [5], abstractions of widgets as clusters of alternative representations along with methods for selecting among them. Furthermore, any specific manifestation of a metawidget is inherently ephemeral, meaning that as time passes, the appearance will change. MetaWidgets can handle these temporal semantics as part of the object state.

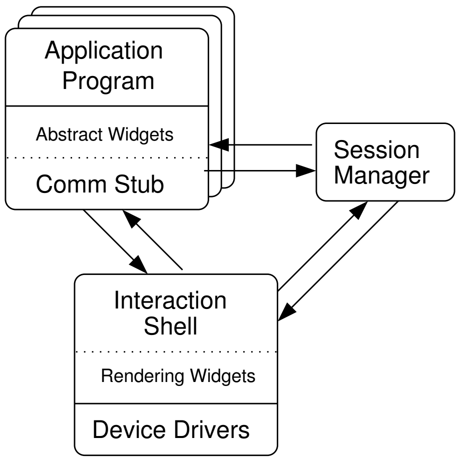
The use of abstract user interfaces is largely based on the observation that application UIs and World Wide Web forms are very similar. Barnicle [2], Pontelli et al. [50], and Theofanos and Redish [63] all researched the obstacles that blind users face when dealing with user interaction models, confirming this observation.

The Views system described by Bishop and Horspool [4] introduces the concept of runtime creation of the user interface representation based on an XML specification of the UI. Independently, Stefan Kost also developed a system to generate user interface representations dynamically based on an abstract UI description [34]. His thesis focussed on the modality-independent aspect of the AUI description, providing a choice of presentation toolkit for a given application. His work touches briefly on the topic of multiple interface representations, offering some ideas for future work in this area, while identifying it as an area of interest that faces some significant unresolved issues.

Mir Farooq Ali conducted research at Virginia Tech on building multi-platform user interfaces using UIML [1]. By means of a multi-step annotation and transformation process, an abstract UI description is used to create a platform-specific UI in UIML. This process takes place during the development phase, as opposed to the runtime processing proposed in this work. His thesis identifies the applicability of multi-platform user interfaces as a possible solution for providing accessible user interfaces, yet this idea was not explored any further beyond references to related work. The research into the construction of accessible interfaces using this approach is left as future work.

User interface description languages (UIDL) have been researched extensively throughout the past eight to ten years. Souchon and Vanderdonckt [60] reviewed different XML-compliant UIDLs, finding that no single UIDL satisfies their requirements for developing fully functional UIs. Trewin, Zimmermann, and Vanderheiden [64,65] present technical requirements for abstract user interface descriptions based on Universal Access and "Design-for-All" principles, and they evaluated four different UIDLs based on those requirements. The authors noted that further analysis at a more detailed level is required in order to provide a realistic assessment.

The Belgian Laboratory of Computer-Human Interaction (BCHI) at the Université Catholique de Louvain developed an UIDL to surpass all others in terms of goals for functionality, "capturing the essential properties [. . .] that turn out to be vital for specifying, describing, designing, and developing [. . .] UIs": UsiXML [41, p. 55]. Of special importance are:



(Image based on [31].)

Fig. 4 The Fruit system

- The UI design should be independent of any modality of interaction.
- It should support the integration of all models used during UI development (context of use, user, platform, environment, . . .).
- It should be possible to express explicit mappings between models and elements.

Building on the growing interest in AUI descriptions, Draheim et al. introduced the concept of "GUIs as documents" [17]. The authors provide a detailed comparison of four GUI development paradigms, proposing a *document-oriented* GUI paradigm where editing of the graphical user interface can take place at application runtime. In the discussion of the *document-based* GUI paradigm, they write about the separation of GUI and program logic [17, p. 70]: "This makes it possible to have different GUIs for different kinds of users, e.g. special GUIs for users with disabilities or GUIs in different languages. Consequently, this approach inherently offers solutions for accessibility and internationalisation." The idea did not get developed further, however.

4.1 Fruit

Kawai et al. describe the architecture for a user interface toolkit that supports dynamic selectable modality, named Fruit [31]. The system accommodates the needs of users with disabilities and users in special environments by means of a model of semantic abstraction, where a separation of concern is enforced by decoupling the user interface from the application functionality. This allows users to select a UI representation that fits their circumstances as much as possible.

The Fruit system has two immediate goals:

– Allow a user who is operating an application program to suspend his or her interactive session, and to resume it from another computer system.
– Allow for the interaction with an application program to switch from an auditory/tactile interface to a graphical interface without interrupting the execution of the application.

In this system, the application program is developed using a UI toolkit that provides abstract widgets, i.e., widgets that define functionality rather than the representation in a specific output modality. The rendering of the UI is delegated to an interaction shell that uses reified widgets, i.e., widgets that inherit from the corresponding abstract widget, and provide added functionality for the rendering of the widget in a specific modality.

4.1.1 Architecture

Fig. 4 shows the architecture of the Fruit system. Essentially, three main components can be identified:

– *Communication stub*: This is a library of abstract widgets to be linked with the application code. Effectively, the application's UI is designed and developed based on this toolkit library, equivalent to how it would be done based on a graphical toolkit. The abstract widgets implement the semantics of user interaction.
– *Interaction shell*: All interaction with the user is handled through an interaction shell. It provides input and output facilities through one or more modalities. In general, a user will use a single interaction shell to operate all his or her applications, though it should be possible to use multiple interaction shells simultaneously. The rendered widgets provide the representation of the UI in specific modalities.
– *Session manager*: This component manages and coordinates the association between the communication stubs of applications and interaction shells. Applications register themselves with the session manager, and the user is then able to "connect" to an application by means of their interaction shell of choice. It also supports suspend/resume operations to allow for switching between interaction shells.

Typical operation starts with a session manager running on a specific host[6], and the user starting an interaction shell of their choice on their system[7]. To invoke a new application, the user uses the interaction shell to signal the appropriate session manager. The session manager handles the launching of the application, providing it with communication parameters so it can contact its controlling interaction shell.

[6] Each host where application code may be executed must have a session manager running.
[7] It is possible for all components to run on a single system.

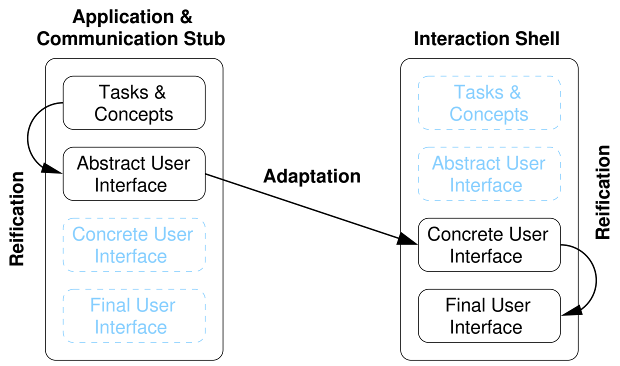


Fig. 5 URF diagram for Fruit

The communication stub initiates contact with the interaction shell, and receives a reply that indicates the capabilities of the shell, e.g., whether it can process bitmapped images or whether it support a pointer device. The communication stub uses this information to filter the information that is sent to the interaction shell, so that only relevant information is transmitted.

User interaction takes place from this point forward as input is passed from the interaction shell to the application, and output is passed back. At any moment, the user can disconnect from the application, causing it to become suspended. User interaction can be reestablished from the same system or from another system, and with either the same interaction shell or a different one.

4.1.2 Analysis

Within the context of the Unifying Reference Framework, the Fruit system can be described as shown in Fig. 5. On the left-hand side, the application-based UI design is shown. The user interface is developed in the common procedural manner, implementing it based on a UI representation toolkit. In the Fruit system, the toolkit is modality independent, providing abstract interaction objects (AIO). This design can be presented as a reification of the tasks and concepts specification into an abstract UI.

The actual representation of the UI is handled by the interaction shell. In terms of the Unifying Reference Framework, this component provides reification from a (possibly filtered) abstract UI to a concrete UI for a specific modality (or set thereof), and further reification into the final UI that it presented to the user for interaction with the application.

The Fruit system satisfies the Equivalence CARE property for output modalities by design, because each and every interaction shell must be capable of providing user interaction for all applications. By implication, Fruit therefore provides dynamic coherence between modalities. Static coherence is also part of the design, because rendering widgets are reifications of abstract widgets in a 1-to-1 relation. However,

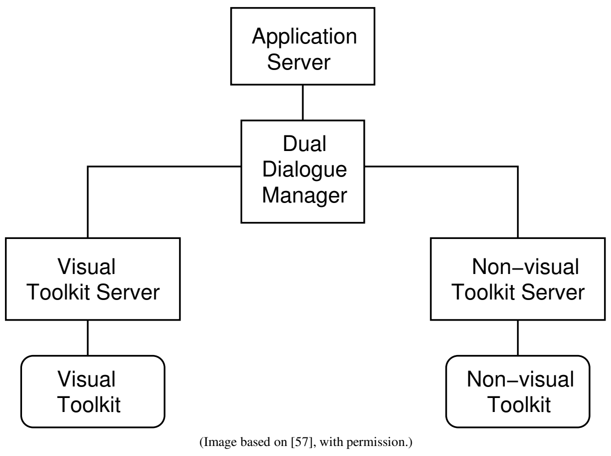


Fig. 6 HOMER UIMS

in its implementation, the system does not adhere to the separation of concerns concept. The communication stub filters the information sent to the interaction shell based on negotiated capabilities, thereby performing modality-dependent processing where static coherence may be lost. It is therefore prudent to consider the static coherence only partially satisfied. Information filtering impacts the ability to convey meaningful graphical information in a non-visual representation as well, yielding only partial support for this aspect of non-visual access to user interfaces.

The interaction shell provides all input and output facilities for a specific modality (or set of modalities), and is responsible for providing the user with a user interaction environment that is equivalent to any other shells in the Fruit system. In order to ensure that all users can access the system in an equivalent way, the interaction shells must all implement appropriate support for exploration and interaction.

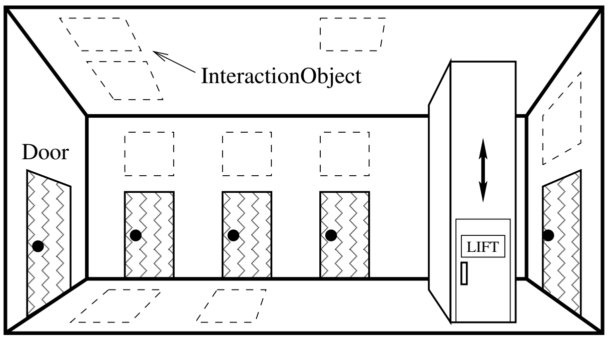
The handling of input modalities is not discussed at a sufficient level of detail in published papers for a determination to be made concerning the CARE properties of this component.

4.2 HOMER UIMS

The HOMER UIMS [56,57] is a language-based development framework, aimed at dual interface development: equal user interaction support for blind and sighted people. The dual interface concept is designed to satisfy the following properties:

1. *Concurrently accessible by blind and sighted people*
   Cooperation between a blind and a sighted user is recognised as quite important to avoid segregation of blind individuals in their work environment. The HOMER system supports cooperation in two distinct modes: both users working side-by-side on the same computer system, or the users (not physically close to one another) working on their own computer systems.
2. *The visual and non-visual metaphors of interaction meet the specific needs of their respective user group, and they need not be the same.*
   This property expresses a potential need for interaction metaphors that are designed specifically for the blind. It is important to note that the HOMER UIMS design is in part addressing the perceived notion that the underlying spatial metaphor for the GUI system is by design based on visually oriented concepts, and therefore not appropriate for non-visual interaction. This property is as such not limited to just metaphors that relate to the visualisation of the UI.
3. *The visual and non-visual syntactic and lexical structure meet the specific needs of their respective user group, and they need not be the same.*
   This property (also in view of the exact meaning of the previous property) addresses the possible need for a separate non-visual interface design. The explicit mentioning of the syntactic and lexical structure in this requirement establishes the scope for the non-visual design proposed here as limited to the perceptual layer, although that is not specifically stated.
4. *At all times, the same semantic functionality is to be accessible to each user group through their respective modalities.*
   Essentially, the underlying functionality should be accessible to all users, albeit possibly through alternative modalities. Regardless of the modality, the functionality must be presented to all users in an equivalent manner.
5. *At all times, the same semantic information is to be accessible to each user group through their respective modalities.*
   Similar to the handling of functionality as expressed in the point above, information should be made available to all users, by means of any appropriate modality that can guarantee equivalency.

The second and third properties are largely based on an analysis of sample user interfaces that employ highly visual idioms. The authors reach the very valid conclusion that the visual user interface may visualise information in a manner that is not accessible to blind users and likewise interaction techniques can be used that are inaccessible. Under those circumstances, it seems logical that a non-visual user interface (NVUI) would be designed with distinct non-visual features (metaphors and design) to provide an accessible solution. The fourth and fifth properties provide further requirements for the visual and non-visual designs.



(Image based on [56], with permission.)

Fig. 7 The “Rooms” metaphor

4.2.1 Architecture

Fig. 6 provides a schematic overview of the “Dual Run-Time Model” introduced with HOMER UIMS [56,57]. The application server provides the program functionality for the system, and remains separate from the user interaction handling. All UI processing originates from the dual dialogue manager, where virtual interaction objects provide the underlying semantics of the user interface. An application programming interface channels semantic operational data between the application and the dual dialogue manager in order to maintain a separation of concerns. The actual realisation of the dual UI presentations is handled by means of an instantiation mechanism within the dual dialogue manager, where the visual and non-visual physical interaction objects that are associated with virtual interaction objects are created. These physical objects are rendered according to specific representation toolkits (as appropriate for chosen modalities) by means of the toolkit servers.

It is important to note that the HOMER UIMS does not require that every virtual interaction object is represented by a visual and a non-visual physical object. Likewise, even when dual physical interaction objects do exist, they need only implement those aspects of behaviour of the virtual object that are relevant for the modality in which the physical object is represented. A common use can be found in various visual effects that are associated with user interaction in a graphical user interface.

In view of the need for a non-visual representation of the user interface, Savidis and Stephanidis developed a new metaphor for non-visual interaction [55], the “Rooms” metaphor. The rationale for this new development can be found in their observation that existing approaches were “merely non-visual reproductions of interactive software designed for sighted users”, and that these approaches “explicitly employ the Desktop metaphor for non-visual interaction.” The alternative presented by the authors is shown schematically in Fig. 7. The interaction space comprises a collection of virtual rooms, where each room acts as container for interaction objects:

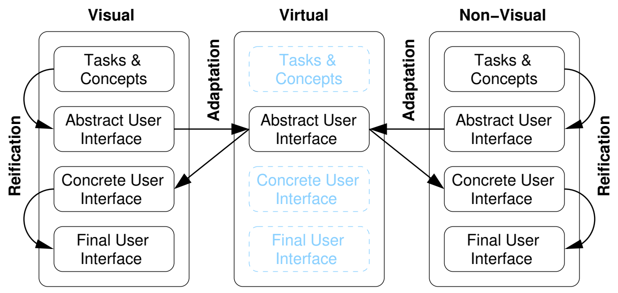


Fig. 8 URF diagram for HOMER UIMS

- *Door*: A door is a portal to a neighbouring room at the same (vertical) level.
- *Lift*: A lift provides access to the room directly above or below the current room, allowing for a change in (vertical) level while retaining the same position on the horizontal plane.
- *Switch*: An on/off switch represents a toggle.
- *Book*: A book is a read-only text entity.
- *Button*: A control to activate some functionality.
- ...

Interaction objects can be placed on any of the six surfaces of the room: the four walls, the floor, and the ceiling. The user is conceptually floating at the centre of the room.

4.2.2 Analysis

Within the context of the Unifying Reference Framework, the HOMER UIMS can be described as shown in Fig. 8. This is a rather unusual design flow, because it does not quite follow the typical model of adaptation from context A to context B as shown in Fig. 3. The UI is designed from two different angles: visual and non-visual. The modality-based designs share a common tasks specification, but they employ distinctly different concept spaces. Reification on both sides yields abstract UI specifications for what is called ’physical interaction objects’ in the HOMER UIMS. These objects are not quite realized UI objects at the concrete level, but rather modality-aware abstractions.

The physical interaction objects from both the visual and the non-visual contexts are combined into a generalized abstraction in the context of modality-independent virtual interaction objects. This is an unsual adaptation step, due to the fact that two distinctly different contexts are combined (and further abstracted) into a single virtual context. The virtual interaction objects defined here are responsible for interfacing with the application.

The realization of the user interface in dual form is accomplished by means of dual reification from the virtual AUI to a visual CUI and a non-visual CUI. The actual presentation to the user is performed by a final reification from the concrete level to the final UI.

Table 1 Comparison between the Fruit system and the HOMER UIMS

		Fruit	HOMER
Coherence	**Static**	Partial	No
	Dynamic	Yes	Yes
Exploration		Yes	Yes
Conveying graphical information		Partial	No
Interaction		Yes	Yes
CARE	**Input**	n/a	AE
	Output	E	AE
Conceptual model		Single	Dual
Concurrency		Partial	Yes

The very design of the HOMER UIMS is based on the notion that non-visual access should be accomplished by means of a new metaphor for non-visual interaction. This makes it essentially impossible to ensure static coherence between the visual and the non-visual representations. Likewise, this system does not ensure that it is possible to convey meaningful graphical information in non-visual form because the non-visual representation of the UI operates in a totally different conceptual model and metaphor.

Dynamic coherence can be guaranteed by the design of this system because regardless of the interaction metaphors used, the Equivalence CARE property applies based on the dual interface concept properties presented on page 8 (properties 4 and 5). The concept properties also provide a basis for being able to ensure that non-visual exploration and interaction are guaranteed.

The dual interface design is not merely an aspect of the representation, i.e., the output modalities. It also affects input handling. For all intents and purposes, the visual and non-visual UIs are independent interfaces that interact with the application by means of a shared central component. The two UI representations handle their own input. From a global point of view, some input may be very specific for just one of the interfaces, whereas the majority of user input will satisfy the Equivalence CARE property for input modalities. Modality specific input handling relates to the Assignment CARE property.

4.3 Comparison

The assessment of the two representative approaches in the context of the requirements for non-visual access to user interfaces and the CARE properties is summarized in Table 1. Neither system provides for all requirements for non-visual access, with the Fruit system not being able to fully guarantee static coherence and the ability to convey graphical information in a non-visual manner. The HOMER UIMS, on the other hand, does not support static coherence and the conveying of graphical information at all because of its design.

Two additional aspects of multimodal user interface design can be considered important within the context of providing both visual and non-visual UI representations:

– *Conceptual model*: This indicates whether the approach is based on a single conceptual model, or on multiple conceptual models. When multiple conceptual models are used, it is very difficult to still be able to ensure coherence.
– *Concurrency*: This indicates whether UI representations can be rendered simultaneously.

The Fruit system is based on a single conceptual model, encoded in the design of the abstract toolkit that is used for the development of the application UI. The HOMER UIMS, on the other hand, presents two conceptual models: one for the visual representation, and one for the non-visual representation.

In terms of concurrency of representations, the HOMER UIMS provides for full concurrency between the visual and non-visual interfaces. This was in fact one of the primary goals of the system. The Fruit system on the other hand does support concurrent representations of a single UI, but due to a lack of detailed information beyond a basic indication that it is possible, it can only be deemed partial concurrency. Amongst the areas of concern are: lack of clarity on how concurrent user input is handled, whether multiple interaction shells connect directly to the application communication stub, and whether the possible filtering of information transmitted by the communication stub is on a per-shell basis or global.

5 Parallel User Interface Rendering

The Parallel User Interface Rendering approach to providing alternative representations of graphical user interfaces is based on the significant advantages of abstract UIs, described in a sufficiently expressive UIDL. It also builds on the ability to present a UI by means of runtime interpretation of the AUI description. This section introduces the design principles that lay the foundation for this novel technique, and provides details on the actual design.

5.1 Introduction

Commonly used accessibility solutions for supporting blind users on graphical user interfaces (on UNIX-based systems and elsewhere) are mostly still "best-effort" solutions, limited by the accuracy of the off-screen model that they derive from the GUI. The OSM is created based on a variety of information sources. Much of the syntactic information

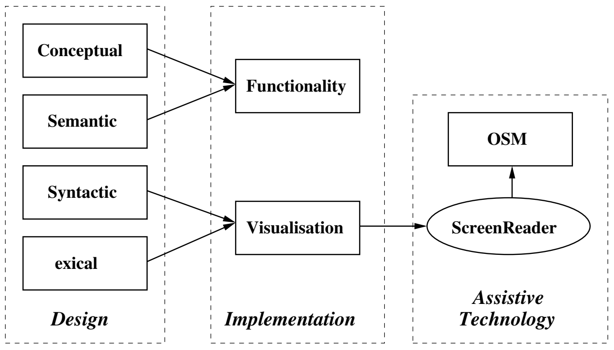


Fig. 9 Screen reader using the visual UI as information source.

can be obtained from hooks in the graphical toolkit, and by means of support functions in the graphical environment. In addition, various toolkits provide an API for AT solutions to query specific information about the GUI. Sometimes, more advanced techniques are necessary, such as OCR and interposing drivers to capture data streams for low-level analysis [25]. Further enhancement of the OSM information often involves complex heuristics and application-specific scripting.

Fig. 9 provides a schematic overview of a screen reader implementation that is typically used in current AT solutions. As described above, the screen reader derives an off-screen model from the visual representation of the GUI[8]. The visualisation part of the implementation is created based on the syntactic and lexical portions of the UI design, and therefore the screen reader is unable to access important semantic information about the UI [9]. This has proven to be a significant limitation, often requiring application specific scripting on the side of the screen reader to essentially augment the OSM with semantic information. Obviously, this is a less than ideal solution because it requires UI data to be maintained outside of the actual application. Accessibility is also limited in function of the ability of the script writer to capture application semantics accurately.

Clearly, OSM-based screen readers still operate entirely based on information from the perceptual layer, and they are thereby limited to providing a translated reproduction of the visual interface. This is an improvement over the traditional approach of interpreting the graphical screen, but it still depends on strictly visual information, or an interpretation thereof. The complications related to this approach are reminiscent of a variation of the "Telephone" game, where a chain of people pass on a message by whispering from one to the next. Only, in this case the first person (Designer) describes (in English) a fairly complex thought to the second person (Implementation), who explains the thought to the third person (Visual Presentation), who writes down the actual information and passes it to the fourth person (Screen Reader, who does not read English fluently), who then verbally provides the last person (a blind user) with a translation of the message. The probability that the message passes through this chain without any loss of content is essentially infinitesimal.

[8] Even though some of the data capturing may take place between the application and the graphical toolkit (e.g., by means of an interposing library), and therefore prior to the graphical rendering, the data can still be considered visual because the application usually either tailors the data in function of the chosen representation, or it passes it to specific functions based on a chosen visualisation.

Various research efforts have focused on this problem throughout the past 10–15 years (e.g., [57,7,35]) with mixed success and often quite different goals.

Parallel User Interface Rendering is a novel approach based on the following fundamental design principles:

1. A consistent conceptual model with familiar manipulatives as basis for all representations.
   See section 5.2.
2. Support for multiple toolkits at the perceptual level.
   See section 5.3.
3. Collaboration between sighted and blind users.
   See section 5.4.
4. Multiple equivalent representations.
   See section 5.5.
   (a) The same semantic information and functionality is accessible in each representation, at the same time.
   (b) Each representation provides perceptual metaphors that meet the specific needs of its target population.

This section commences with a discussion of the design principles for principles for Parallel User Interface Rendering, which are then further refined into the actual design.

5.2 A consistent conceptual model with familiar manipulatives as basis for all representations

The conceptual model is by far the most important design principle for any user interface, and therefore lies at the very basis of Parallel User Interface Rendering. Seybold writes (discussing the Star project at Xerox PARC) ([59, p. 246], quoting from [58]):

> "Most system design efforts start with hardware specifications, follow this with a set of functional specifications for the software, then try to figure out a logical user interface and command structure. The Star project started the other way around: the paramount concern was to define a conceptual model of how the user would relate to the system. Hardware and software followed from this."

Smith, et al. define a user's conceptual model as [59]:

> Conceptual Model: The set of concepts a person gradually acquires to explain the behaviour of a system.

Based on this very definition one might conclude that a conceptual model is a user specific model, based on personal experiences while operating or interacting with a system. While that is generally true, ample examples can be found that indicate that there are many "systems" for which a collective conceptual model exists, i.e., a model that is shared by most people. Examples include the operation of household appliances, driving an automobile, etc. In some cases, the same conceptual models may develop independently for multiple individuals as they interact with a system in the same environment, so that their experiences are sufficiently similar. More often, however, a conceptual model becomes a collective conceptual model through teaching, be it direct or through guidance.

Given the need to define a conceptual model for a user interface, designers have essentially two choices: design the user interface based on a existing model employing familiar metaphors, or develop a brand new model[9]. Extensive research was done when the original GUI concept was developed, leading to the conclusion that the metaphor of a physical office is an appropriate model [59]. It is however important to note that this conclusion was reached in function of developing a user interface for visual presentation, and non-visual interaction was therefore not taken into consideration.

Is it then possible to have a consistent conceptual model as basis for both visual and non-visual representations, or does the lack of sight necessitate a specialised non-visual conceptual model?

Savidis and Stephanidis suggest that specific interaction metaphors are to be designed for non-visual interaction, because the "Desktop" metaphor is visual by design [55,57]. This seems to be contradictory to the design principle presented in this section, especially given that Gaver explicitly states that [22, p. 86]: "The desktop metaphor [. . .] is the result of a conceptual mapping.", thereby clearly associating the Desktop metaphor with the conceptual model. Savidis and Stephanidis, however, do not differentiate the multiple layers of metaphors in a user interface the same way as Gaver does, as evidenced by an important statement of scope in their work [55, p. 244]: "The work [. . .] concerns the general metaphor for the interaction environment (i.e. how the user realizes the overall interaction space, like the Desktop metaphor) which is different from metaphors which are embedded in the User Interface design." They associate the Desktop metaphor with the perceptual level instead. Their work is therefore not contradicting the design principle presented here. Furthermore, it actually provides support for the design principle of needs-driven perceptual metaphors for representation, as discussed in section 5.5.1.

Kay presents some compelling arguments to avoid the terminology of metaphors in relation to the conceptual model

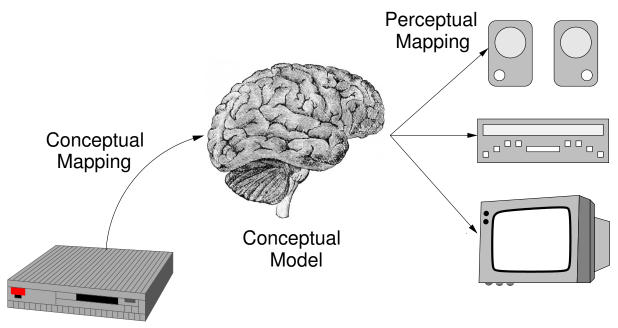


Fig. 10 The role of the conceptual model

[32], instead referring to "user illusion" as a more appropriate phrase. Indeed, conceptual metaphors used in user interface design are more often than not a weak analogy with their physical counterpart. While the notion of a "paper" metaphor is often used within the context of a word processing application, users are not limited by the typical constraints associated with writing on physical paper. Moving an entire paragraph of text from one location to another on the "paper" is quite an easy task on a computer system, whereas it is rather complicated to accomplish on physical paper. Smith, et al. expressed a similar notion when expressing that: "While we want an analogy with the physical world for familiarity, we don't want to limit ourselves to its capabilities." [59, p. 251] It is clear that the underlying conceptual model is therefore a tool to link the internal workings of the computer system to a model that the users can relate to, without requiring the model to be bound by constraints of the physical world, and without any assumptions about representation (see Fig. 10). The mental model reflects the metaphors that link the computer reality[10] to a task domain.

With the relaxed interpretation of metaphors and under the assumption that the separation between the conceptual and perceptual layers is maintained, the real question becomes whether a blind user can fully comprehend the conceptual model for a user interface. At this level, both sighted and blind users are faced with a mental model[11] that captures manipulatives[12]. The common model represents an office or a desktop, concepts that sighted and blind people are certainly familiar with. Whether a user can conjure up a visual image of the mental model is not necessarily relevant in view of the assumed separation from the perceptual. Still, often people will use visual imagery to represent the

[9] This not only involves defining objects and activities (functionality of objects), but also developing strategies to introduce the new model to the anticipated user population.

[10] "Computer reality" is defined by Gaver as [22, p. 85] "the domain in which computer events are described, either by reference to the physical hardware of the system or its operations expressed in some programming language."

[11] Sighted users may often not even realize that a mental model is involved due to the fact that a GUI is generally presented visually using iconic elements that are closely related to the underlying conceptual (mental) model.

[12] Objects and the manipulations that are possible on and with those objects.

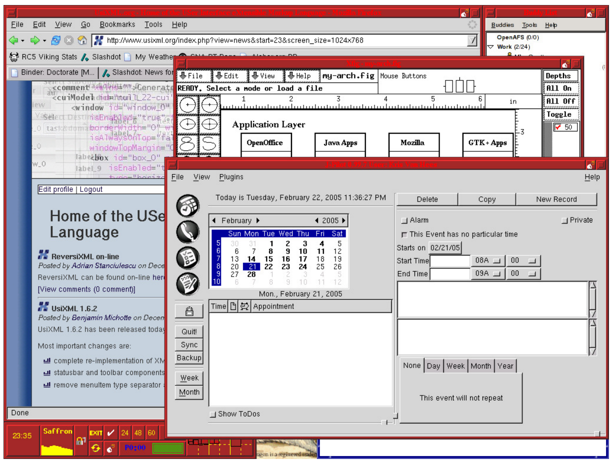
model in their mind, regardless of whether that is truly necessary to reason about it, and this applies to both sighted and many blind users. The exact nature of this mental image and whether it is truly visual or perhaps perceptual in an alternative form is a matter of individual preference and ability. Edwards states in a very insightful yet unpublished paper (in draft) [18]: "If an existing visual interface is to be adapted, it may be that there are aspects of the interface which are so inherently visual that they will be difficult to render in a non-visual form [. . .] but the suggestion is that this need not be the case, if the designer would not commit to a visual representation at an early stage of the design process." This early stage would correspond to the development of the conceptual model for the user interface, which by design should be medium-independent in order to support this level of accessibility [19].

In an inspiring article in The New Yorker, Oliver Sacks wrote about several blind individuals who all experienced the effects of blindness on visual imagery and memory in different ways [52]. One individual effectively lost not only the ability to visualise but in fact the very meaning of visual concepts such as visible characteristics of objects and positional concepts based on visual imagery. Other people reported an enhanced ability to use visual imagery in their daily life. In addition, Sacks also wrote about sighted people who did not possess visual imagery. Noteworthy is that none of the people discussed in Sacks' article seemed to have any difficulties leading a functional and successful life in a predominantly sighted world. Whether one can visualise the physical does not seem to impact one's ability to operate in and interact with the world. Within the context of conceptual mental models, the focus should be on on what objects exist in the model, and what one can do with them, rather than what objects look like or how they function.

The research and analysis presented in this section supports the notion that, when a clear separation between the perceptual and the conceptual is maintained, a single conceptual model for a user interface can be appropriate for both blind and sighted users. There is therefore no need to consider a separate non-visual UI design at the conceptual level.

5.3 Support for multiple toolkits at the perceptual level

Providing access to GUIs for blind users would be relatively easy if one could make the assumption that all applications are developed using a single standard graphical toolkit *and* if that toolkit provides a sufficiently feature-rich API for assistive technology. Unfortunately, this situation is not realistic. While the majority of programs under MS Windows are developed based on a standard toolkit, the provided API still lacks functionality that is necessary to ensure full accessibility of all applications. X Windows does not impose the



File Edit View Go Bookmarks Tools Help
Home of the USer Language
Application Layer
OpenOffice
Java Apps
Mozilla
GTK+ Apps
Today is Tuesday, February 22, 2005 11:36:27 PM
Delete
Copy
New Record
Alarm
Private
This Event has no particular time
Show ToDos
This event will not repeat

Fig. 11 X11 session with multiple graphical toolkits

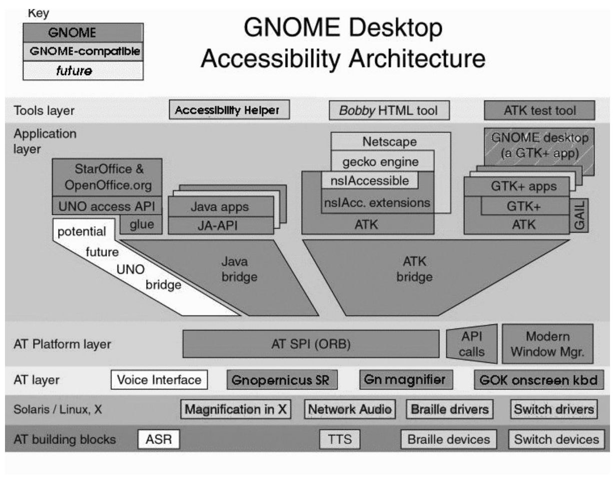
use of any specific toolkit, nor does it necessarily promote one. It is quite common for users of a UNIX-based system to simultaneously use any number of applications that are each built upon a specific graphical toolkit. Some applications even include support for multiple graphical toolkits, providing the user with a configuration choice to select a specific one.

In order to be able to provide access to application user interfaces regardless of the graphical toolkits they are developed against, the chosen approach must ensure that the provision of non-visual access is not only medium-independent but also toolkit-independent.

Fig. 11 shows a fairly typical session on a UNIX system, displaying four applications: Firefox, J-Pilot, Xfig, and GAIM. Firefox and J-Pilot are built upon the GTK 1.2 graphical toolkit. Xfig uses the Athena Widget Set, whereas GAIM uses GTK 2.0. The bottom of the figure also shows the window manager button bar. The "Look & Feel" of the graphical interaction objects for the four applications is quite different, yet sighted users know intuitively how to operate them. All buttons, input fields, and menu bars essentially work the same way, regardless of how they look. To a blind user, the visualisation of the UI element is obviously also irrelevant, unless somehow it is used to convey information to the user[13]. The problem therefore lies with the implementation of the accessibility solution and/or the implementation of the toolkits. Toolkits often have very different ways in how they implement the visualisation of a specific UI element, which complicates the introduction functionality to support AT needs. Toolkit developers (and vendors) may also be less inclined to keep up with AI changes, etc.

In the GNOME Accessibility Architecture the complexity of flexibility is handled in a very explicit manner: toolkits

[13] It is safe to assume for the purpose of this discussion that the element provides user interaction functionality only. Conveying information is an aspect of abstract UI semantics that can be represented in various ways - it is not purely related to visualisation.



(Reprinted from [24] with permission.)

Fig. 12 The GNOME Desktop Accessibility Architecture

are expected to implement an accessibility API that provides an end point for communication with the AT-SPI component by means of a (API specific) bridge (see Fig. 12). Accessibility for OpenOffice.org requires the UNO access API, and communication with AT-SPI is currently handled by means of a Java bridge. Likewise, Java programs make use of the Java Accessibility API, and the Java bridge, whereas GTK-based applications use the GNOME Accessibility ToolKit (ATK), along with a specific ATK bridge to AT-SPI. Only at the AT-SPI level can one consider the solution toolkit independent. In a posting to the GNOME Accessibility mailing list on July 20th, 2004, Peter Korn stated that GNOME was taking the approach that applications must "opt-in" to accessibility. This is generally done by having the application code explicitly call functions in the toolkit's accessibility API to provide information that can be queried later by means of AT-SPI, and limiting oneself to using widgets and features that are well supported in the accessibility architecture. Another aspect often involves an additional UI description that can be loaded by an accessibility component in the toolkit implementation. The downfall of this approach is that the process of keeping the UI description in sync with the application UI code is entirely manual. Note also that there are a significant amount of common applications that are developed as part of a competing graphical desktop environment system (KDE), and those applications are not accessible at all under GNOME because an implementation of a bridge between Qt (the KDE graphical toolkit) and AT-SPI is still pending.

Many of the existing approaches based on an accessibility API involve mapping graphical toolkit features (widgets and their interaction semantics) onto a set of accessible features. This effectively amounts to creating an abstraction from a realized (concrete) user interface, often with an implicit loss of information[14]. When confronted with multiple graphical toolkits, multiple distinct mappings are required to accommodate abstracting all concrete UIs onto a defined model. This has proven to be quite complex, and typically involves significant limitations as shown above.

Section 5.6 will show that when multiple toolkits operate on the same conceptual model, effectively performing a reification of an abstract user interface within the context of a specific "Look & Feel", a unified source of information is available for assistive technologies to operate upon. Mapping is no longer required because accessibility information can be derived directly from the abstraction that previously had to be derived from each concrete UI by means of a specialised bridge.

5.4 Collaboration between sighted and blind users

The Collins English Dictionary – Complete & Unabridged 10th Edition defines the term "collaboration" as follows:

> Collaboration: The act of working with another or others on a joint project.

In order to ensure that segregation of blind users due to accessibility issues can be avoided, appropriate support for collaboration between the two user groups is important. This collaboration can occur in different ways, each with its own impact on the overall requirements for the accessibility of the environment.

Savidis and Stephanidis [57] consider two types of collaboration:

– *Local*: Sighted and blind users interact with the application on the same system, and they are therefore physically near one another. This would typically involve one user approaching the other in order to ask a question or share information about interaction with the application. Common occurrences are often characterised by conversations "let me do . . . ", "let me show you . . . ", and "how about we do . . . ".
– *Remote*: Sighted and blind users access the application from different systems, and they are physically distant from one another. This situation would typically still involve a conversation as illustrated in the previous item, possibly by means of a communication channel outside the scope of the application (e.g., by phone). It is important to note that both users are using physically distant systems to access the application on a central system.

The distinction of these two types is not sufficient to accurately describe the possible ways in which sighted and

[14] It is important to note that although information may be lost, accessibility may not be impacted because often only perceptual information is affected.

blind users may collaborate. Proximity between users[15] does not actually impact collaboration much as long as an adequate communication channel[16] is available. The ability to interact directly with the computer system that the users collaborate about can certainly be a benefit, and this can often be accomplished both locally and by means of remote connectivity. Note that local collaboration between two blind users can be a bit less efficient when a refreshable Braille display is used, because typically only a single device will be supported for a given computer system, requiring a pass-the-keyboard style interaction. The work presented here allows for multiple refreshable Braille displays to be operated simultaneously on a computer system.

Consider a case where a worker walks up to a co-worker and they discuss the user interface of an application. Even without touching the computer system, the users can still talk about the operation of the application, e.g. "Select this and that, and then click on the button that reads ...". This requires that the users can understand the user interaction semantics of the UI elements that are part of the discussion, and that the user can either visualise the user interface, or otherwise reason about it based on an alternative perceptual model.

Also consider a situation where a worker needs to call someone for help, and those who can help are at a remote location. In the current age of distributed computing and multi-location businesses this situation is quite common. In this situation, collaboration may again be limited to a communication channel only.

The communication portion of collaboration can occur at two different levels (applying [22] in the broader context of interaction between users as it relates to GUIs):

- *Conceptual*: Users talk in terms of the semantics of user interaction. A quite typical conversation would resemble: "Select double sided printing, and the collate option, and then print the document." When direct interaction is possible (be it local or remote), the dialogue is likely to be augmented with either a demonstration of the interaction, or a verification that the instruction is understood correctly.
- *Perceptual*: Users talk in terms of how to operate manipulatives that are present in the UI. Conversations would resemble: "Check the double sided check box and the collate check box, and then click on the 'Print' button." Direct interaction would add a demonstration component to the exchange, or alternatively, a mode of verification that the other person is accurately following the directions.

[15] Either between blind users, or between a sighted and a blind user. Sighted users often tend to depend on a visual focal point when collaborating about the interaction with a system or an application.

[16] The main requirement for the communication channel is that it provides for sufficient bandwidth to enable efficient and specific exchange of information. A phone connection is often much more constructive to collaboration than, e.g., an online chat session.

Since proximity between users does not directly impact their collaboration, there is no need to make a distinction between local and remote. On the other hand, the fact that communication between users can take place on two different levels does require two specific cases to be considered.

Communication at the perceptual level involves details about the representation of the user interface in a specific modality. It is therefore not an effective way for sighted users and blind users to discuss the operation of an application. It requires that both parties have a good understanding of the actual operational details of the representation of the UI. While many blind users certainly understand the vast majority of visual concepts [52], manipulation of UI elements in the visual representation is often not possible[17] [57]. It is also important to note that an expectation for blind users to be able to communication with others at the perceptual level amounts to reducing the level of accessibility back to the graphical screen rather than the GUI.

> Perceptual collaboration: The act of working with another or others on a joint project with communication at the perceptual level (i.e., within the context of the representation of the user interface in a specific modality).

Communication at the conceptual level relates directly to the semantics of user interaction for the application, independent from any representation. In this case, the users communicate within the context of the conceptual model that lies at the core of the user interface. Rather than referring to elements of the representation of the UI (visual or non-visual), users refer to elements of the conceptual user interface.

> Conceptual collaboration: The act of working with another or others on a joint project with communication at the conceptual level (i.e., within the context of the modality-independent user interface interaction semantics).

5.5 Multiple equivalent representations

A common problem with existing approaches for non-visual access to GUIs is related to the use of an off-screen model: lack of coherence between the visual and the non-visual interfaces. Mynatt and Weber identified this as one of the important HCI issues concerning non-visual access (see section 3.3).

The problem is most often related to the information gathering process that drives the construction of the OSM[18].

[17] Or at least, not possible in an equivalent and/or efficient manner.

[18] Kochanek provides a detailed description of the construction-process for an off-screen model for a GUI [33].

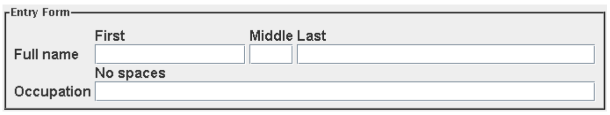
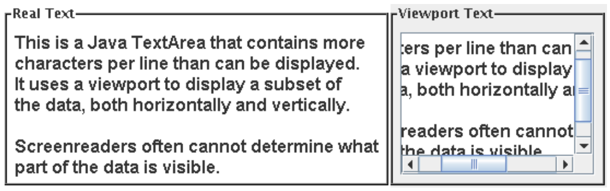


Fig. 13 Example of a visual layout that can confuse screen readers.



This is a Java TextArea that contains more characters per line than can be displayed. It uses a viewport to display a subset of the data, both horizontally and vertically.

Screenreaders often cannot determine what part of the data is visible.

```
ters per line than can
a viewport to display
a, both horizontally a

readers often cannot
the data is visible
```

Fig. 14 Example of the effects of a viewport on text visualisation.

Limitations in being able to obtain the following two pieces of information often lead to this lack of coherence:

– *Semantic relationships between user interface elements.* A typical example can be found in the common relation between label and input fields. If this relation is either not encoded in the GUI implementation, or if it is not available through hooks, the OSM will contain the label and the input field as independent elements. The screen reader will present them as such, which may cause the user to be presented with an input field without any indication to what data is expected to be entered there. On screens where multiple input fields occur in close proximity, this can result in significant levels of confusion and potential data entry errors. Fig. 13 shows an example of how lack of semantic relation information can confuse a screen reader. The placement of multiple label widgets in close proximity to a text entry field makes it difficult to determine what label should be spoken when the user accesses the text entry field.
– *Effects of the visualisation process on the actual data contained in user interface elements.* This problem is mostly observed with larger text areas, where the text that is actually visible in the GUI may be much less than the actual text that is contained within the UI element. GUI toolkits often use a viewport-technique to render a subset of the information in the screen, in function of the available space on the screen (Fig. 14). If the assistive technology solution (the screen reader) cannot obtain information on what portion of the text is visible to the user, a blind user may be presented with data that a sighted colleague cannot see on the screen. In Fig. 14, a blind user would typically be presented with an equivalent of the view at the left whereas a sighted user would be presented with the (more limited) view on the right. This leads to significant difficulties if the two users wish to collaborate concerning the operation of the application and the content of the text area.

Both problems can be solved by providing a single user interface representation that is tailored to the specific needs of a user or target group. Coherence is in that case not an issue, because only one interface is ever presented at any given time. Unfortunately, this impacts collaboration between users with different needs negatively, because not everyone will be able to accurately determine the state of the user interface at any given time. Limiting the user interaction to just one target population at a time is cleanly not an acceptable solution.

The discussion on the collaboration design principle (section 5.4) shows that direct access to the system is fundamental to working together successfully. The requirements for making that possible can be summarised as follows:

– *Users can access the system concurrently.* This requirement has been discussed in the preceding paragraphs.
– *Users can interact with the system by means of metaphors that meet their specific needs.* While the user interface is designed using a single consistent conceptual model with familiar manipulatives (see section 5.2), the actual representation that the user interacts with should meet the specific needs of that user. This relates directly to the perceptual layer, covering both the lexical and syntactic aspects of the UI design, and the perceptual metaphors of interaction as appropriate for the needs of a specific user population. Mynatt, Weber, and Gunzenhäuser [46,23] present HCI concerns related to non-visual representations of GUIs, identifying the need to support exploration and interaction in a non-visual interface. The generalised interpretation of this concern is captured by this requirement, and is discussed further in the remainder of this section.
– *Coherence between UI representations is assured.* The preceding two requirements effectively describe a configuration where multiple users can access the system concurrently by means of representations that meet their individual needs. The HCI concerns phrased by Mynatt, Weber, and Gunzenhäuser [46,23] relate to a more specialised scenario of a dual representation (visual and non-visual). Coherence between the representations was presented as an important concern. In the more general context of multiple concurrent representations, the need for coherence certainly remains. Further discussion on this requirement is presented at the end of this section.

5.5.1 Each representation provides perceptual metaphors that meet the specific needs of its target population

At the conceptual level, all users are presented with the same user interface. No adaptations are necessary because the conceptual model is by design not dependent upon any modality of interaction (see section 5.2). Edwards suggested that there

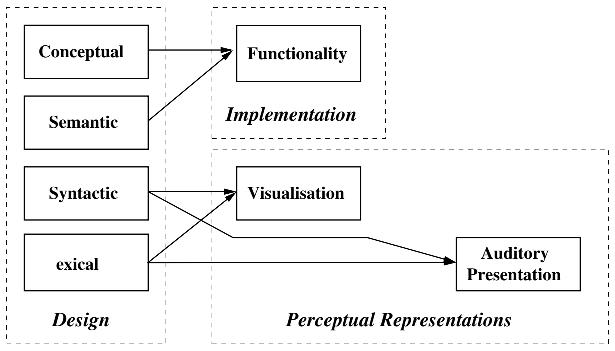


Fig. 15 Multiple perceptual representations of the same user interface *Concurrent representations of the same user interface, each providing perceptual metaphors that meet the specific needs of a group of users.*

is not really a substantial difference between the conceptual models of blind users and sighted users, but rather that the information channels used are different [18]. Extensive research has been conducted in exploring the information channels that are most conducive to perceptual reasoning in blind individuals ([22,45,10,54,53,19]). At the perceptual level, tactile and/or auditory presentation of information is by far the most appropriate for non-visual access. It is therefore obvious that blind users would be best served with specific non-visual metaphors of interaction. Likewise, other groups of users with specific needs may benefit from specific metaphors of interaction that are different from those that are provided in, e.g., the visual representation of the UI.

Fig. 15 illustrates the design principle of specialised representations at the perceptual layer. The conceptual layer captures both the conceptual and semantic levels of the design, providing a description of the functionality of the application. The lexical and syntactic levels of the design need to be interpreted within the context of the target population[19], thereby driving the design of the perceptual layer. The presentation of the UI can essentially become a pluggable component[20], interfacing with the conceptual layer of the UI.

5.5.2 The same semantic information and functionality is concurrently accessible in each representation

Edwards, Mynatt, and Stockton define the semantic interpretation of the user interface as [20, p. 49] “the operators, which the on-screen objects allow us to perform, not the objects themselves.” This refers to the actual functionality that an application provides, but is only part of the semantic layer in UI design. Trewin, Zimmermann, and Vanderheiden provide a more extensive list of the core elements of a user interface based on what they believe should be represented in an abstract user interface description[21] [64]: variables (modifiable data items) that are manipulated by the user, commands the user may issue, and information elements (output-only data items) that are to be presented to the user.

Therefore, the semantic model of the user interface consists of:

> Semantic Information: Data elements in the UI that have meaning within the context of the conceptual model.
>
> Semantic Functionality: Operations that can be performed in the UI and that have meaning within the context of the conceptual model.

The semantic information covers both dynamic data that is encapsulated in UI elements that allow the user to input or manipulate that data, and static data that carries meaning and that is to be presented to the user. It also captures those properties of data items that associate additional meaning to the data.

The semantic functionality captures the manipulations that are possible for each UI element, regardless of representation. The functionality is defined in context of the tasks that can be accomplished with the application.

In order to ensure coherence between concurrent representations of the UI, the same semantic information and functionality must be used to render the UI for each user based on their needs. All users must be able to perform the same user interaction operations at a semantic level, and all users must be able to observe all results of any such operation at the same time[22]. While this seems to be a rather obvious requirement, existing approaches to providing access to GUIs for the blind often provide a sub-optimal implementation where the sighted user is able to interact with a UI element prior to a blind user being able to observe (through synthetic speech or Braille output) that the element exists. Similarly, sometimes a blind user can interact with a UI element that isn’t actually visible to a sighted user. This is an example of semantic functionality that is accessible in one representation but not in another.

5.6 Design

Using the design principles presented in the previous sections, a framework can be designed for providing access to

[19] Specifically, the needs of the target group as those relate to UI interaction.

[20] A pluggable component is one that can easily be replaced by an equivalent component. The term is commonly used in UI contexts to indicate exchangeable presentation components. It is a derivative of the “plug-n-play” hardware concept.

[21] This AUI description effectively defines the user interface at the conceptual and semantic level. It is a formal description of the conceptual mode.

[22] Note that this does not necessarily imply that the result of user interaction is immediate, although it has become common practice to provide near-immediate results in support of the WYSIWYG design principle.

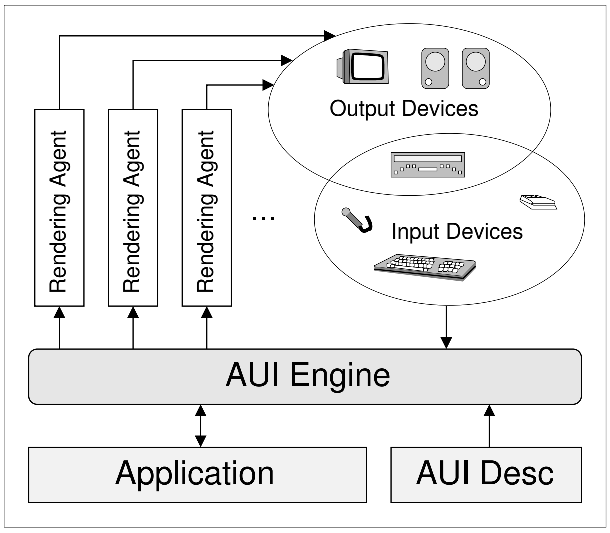


Fig. 16 Schematic overview of Parallel User Interface Rendering

graphical user interface for blind users. This section provides details on the various components.

The schematic overview of the Parallel User Interface Rendering approach is shown in Fig. 16. Rather than constructing the UI programmatically with application code that executes function calls into a specific graphical toolkit, applications provide an AUI description expressed in a UIDL. This authoritative AUI description is processed by the AUI engine, and a runtime model of the UI is constructed. The application can interact with the AUI engine to provide data items for UI elements (e.g., text to display in a dialog), to query data items from them (e.g., user input from a text input field), or to make runtime changes in the structure of the UI. The AUI engine implements all application semantics, ensuring that the functionality does not depend on any specific modality.

The representation of the UI is delegated to modality specific rendering agents, using the UI model at their source of information. At this level, the AUI is translated into a concrete UI (CUI), and the appropriate widget toolkit (typically provided by the system) is used to present the user with the final UI, by means of specific output devices. Therefore, the UI model that is constructed by the AUI engine serves as information source for all the different rendering agents. All representations of the UI are created equally, rather than one being a derivative of another[23]. The application cannot interact with the rendering agents directly, enforcing a strict separation between application logic and UI representation.

[23] It is important to note that it is not a requirement that all representations are generated at runtime, although development time construction of any representations could imply that dynamic updates to the UI structure are not possible.

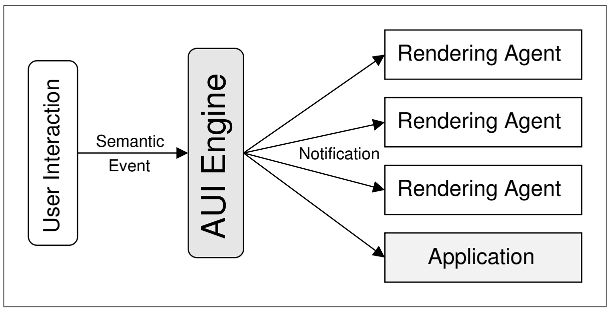


Fig. 17 Logical flow from interaction to presentation

The handling of user interaction events from input devices[24] occurs at the AUI engine level. The PUIR framework is based on meaningful user interaction, and therefore only semantic user interaction events are given any consideration. Given that events are typically presented to toolkits by means of OS level device drivers, and the fact that these event sources are very generic[25], additional processing is required in order for the PUIR framework to receive the semantic events it depends on.

Fig. 17 shows the flow of user interaction through the PUIR framework. User interaction is presented to the AUI engine as semantic events (activate element, select element, etc.) Processing of the semantic event results in a notification event being sent to all rendering agents and the application. This event indicates that the semantic operation has completed. Rendering agents will use this event to trigger a presentation change (if applicable) to reflect the fact that a specific semantic operation took place on a specific widget. The application may use the notification to determine whether program logic must be executed as a result of the semantic operation, e.g., activating the submit button for a form might trigger validation and processing of the form content.

5.7 Conceptual model

The most fundamental design principle is the establishment of a conceptual model as the basis for all UI representations. Discussion of this principle in section 5.2 not only shows that a single model suffices, but also that the well established metaphors of the physical office and the desktop may be appropriate for both blind and sighted users.

It can be argued that these conceptual models are inherently visual, based on a spatial metaphor from a visual perspective, and that it is therefore advisable to design a specialised non-visual model [55]. This argument, in and of

[24] The physical devices that the user employs to perform operations of user interaction with the application.

[25] Device drivers at the OS level are meant to serve all possible consumers. The events they generate are most commonly very low-level events.

itself, is not sufficient to dismiss the existing models and metaphors as not appropriate for blind users, because it makes the assumption that they are not appropriate for the blind just because they have been selected based on visual considerations. The argument also fails to take collaboration between users with different abilities into consideration, and this is another important design principle.

To illustrate the problems with the aforementioned argument, consider the case of print-to-Braille transcription. Braille books are known to be quite bulky due to the nature of Braille (standard cell size, and limited opportunities to represent a sequence of characters with a shorter sequence of Braille cells[26]) and it is therefore more convenient to transcribe books in ways that promote minimising the number of pages needed to transcribe the print text. However, doing so would make it much more difficult to collaborate with sighted peers because correspondence between, e.g., page numbers is lost completely. It is for this reason that the Braille Association of North America stipulate that with the exception of some preliminary pages, all pages of text must be numbered as in the print book [8]. Furthermore, it specifies strict rules on how to indicate print page number changes in the middle of a Braille page, to facilitate collaboration and to support page-based references to print materials. This is a clear example where the need to support collaboration between sighted and blind users outweighs the advantages of a format that is optimised for blind users, without any regard for being able to consult references.

Blind users live and are taught in a predominantly sighted world, where they learn to interact with many of the same objects and concepts as their sighted peers. While there are often definite differences in user interaction *techniques* between the two groups[27], the semantics of the manipulation are essentially the same. As an example, consider the task to lower the volume of a music player. Regardless of whether the control to do so is a slider or a turn knob, once users know where a control is located, they generally know how to operate it[28].

It is also important to observe that although the established terminology seems to refer to visual aspects, the underlying concept is often more abstract. One does not generally consider a GUI "window" in any way equivalent to a physical window, but rather it is seen as a two-dimensional area that usually contains other UI elements. In a sense, it is a top level grouping of all or part of an application UI. Many people are able to identify windows on a screen because they have been taught that a certain entity on the screen is called a "window", but it could as easily have been named a "tableau". What really matters is what you can do with it (semantics).

Table 2 Examples of controls and objects in the conceptual model

Control/Object	Description
Window	Logical container for controls, establishing a context for focusing user interaction.
Button	Control that can be activated, and that can either automatically reset to its default state after activation, or that operates as a switch between two states.
Set of "radio" buttons	Set of buttons where only a single one can ever be selected. It remains selected until another button in the set is selected.
Slide control	A control that allows the user to select a setting from a discrete range of values, e.g., a volume setting on a radio.
Input field	A place for the user to enter information.
Label	An object that conveys information to the user, e.g., a description of a control.
Selection list	A control that allows the user to select one or possibly more items in a list., e.g., a checklist for a complex task.

Similarly, a "button" is rarely interpreted as a strict analogy with physical push button controls. Instead, it is intuitively accepted as a UI element that triggers something when it is selected. In general, most users have learnt to think about UI elements in terms of their semantics and less in terms of what they might represent in the physical world.

The various controls that are represented in the conceptual model for the UI (see Table 2 for examples of controls and objects) are somewhat weak analogies for their physical counterparts. As Kay stated, they should perhaps be referred to as "user illusions" [32]. Both sighted and blind users must approach them as concepts that are familiar, yet the mode of interaction is inherently different. Neither user group can truly push a button or move a slider that is merely presented as a visual image or an auditory artifact. Still, users will communicate in terms of the metaphor because of the familiarity of the concept.

The PUIR framework is designed around this very notion, allowing for a single conceptual model that is not only familiar to all users, but that has also been a well established model within the context of computer systems for many years. Using the physical office and desktop analogy as underlying model for this work also helps maximise the opportunity for collaboration between users with differing abilities.

[26] This is known as a contraction in English Braille, American Edition.
[27] It is obvious that even amongst the blind or the sighted, not necessarily everyone will prefer everything the same way. This has been a driving force behind the efforts to provide user customisations for UIs.
[28] Generically, this type of control is known as a "valuator".

5.8 AUI descriptions

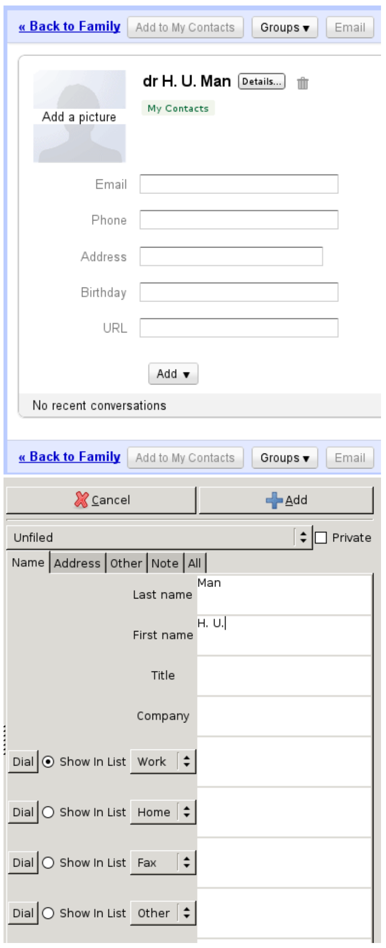
Edwards, Mynatt, and Stockton suggested that the best approach for creating non-visual user interface representations is a translation of the UI at the semantic level, i.e., render the UI in function of the needs of the user, using perceptual metaphors that meet the needs of the target population (see section 5.5.1, page 16). The perceptual layer is merely a reification of an application's abstract user interface. Their goal was to provide non-visual access to existing X11-based applications, and their work therefore involved attempting to capture application information at the semantic level by means of toolkit hooks. A hierarchical off-screen model was constructed based on this information: a semantic OSM. While this effectively results in a pseudo-AUI description of the application user interface, it is sub-optimal because it is created as a derivative of the programmatic constructs that realize the visual representation.

The concept of abstract user interface descriptions has been known for a long long time already, and it has mainly been intended as a source document for the automated generation of the UI, i.e., generating application source code for the programmatic construction of the UI representation and its interaction with the process logic [39,41]. Using the AUI as the basis for the UI is a powerful concept because it provides a canonical description of the conceptual model, thereby taking a very prominent place in the design process. Given a sufficiently expressive UIDL, the AUI description can enable application designers and/or developers to implement a true separation of presentation and logic, and it can alleviate part of the burden of implementing a UI by providing for its automation.

Even when the UI implementation is generated automatically based on an AUI description, providing non-visual access to the UI is still sub-optimal because information can still only be obtained from the realized graphical user interface.

A possible solution could be to make the AUI description available alongside the application, to be used as an information source in support of AT solutions (i.e., the Glade project [14]). Because the implementation of the UI is still generally hard coded in the application, this approach does open up the possibility that inconsistencies between the application and the UI description occur[29]. In support of the coherence design principle (see section 5.5.2, page 17), the PUIR framework is based on the concept that all representations are to be reified from the same AUI. This is a significant paradigm shift from the majority of AT solutions that are still implemented as a derivative of the GUI.

Can the construction of the UI representation(s) be delayed until execution time of the application? In other words,



Man

H. U.

Fig. 18 Web forms bear a striking resemblance to UI data entry screens.
On the left a web form is shown for a address book contact entry. On the right one can see the UI for an application that implements address book management features.

can the UI be rendered by means of AUI runtime interpretation rather than AUI development-time compilation?

When comparing a form on a web page with an application UI where data entry is expected to occur (see Fig. 18), striking similarities can be observed. Both feature almost identical UI elements: buttons, drop-down lists, text entry fields, and labels. In addition, the obstacles that blind users face when using web forms [50,63] are known to be very similar to the obstacles they face when interacting with GUIs [2]. Furthermore, HTML documents are essentially abstract descriptions, although specific modality dependent information can be embedded in the document as augmentation to the abstract description. Web browsers handle the rendering of the HTML document, providing the user with a representation that is commonly tailored to the device the user is using, and possibly other user preferences. Based on these observations and research on the use of AUI descriptions (such as the works of Bishop and Horspool [4], and of Stefan Kost [34]), it can be concluded that it is possible to de-

[29] This is a common problem in any circumstance where essentially the same information is presented in tow different locations.

scribe user interfaces by means of HTML-style documents, to be rendered in function of the output modalities.

This paradigm shift from representing the UI by means of program code in the application to utilising a system that interprets and renders the UI based on an AUI description document has slowly been taking place for the past ten to twelve years. Yet, the shift has not progressed much past the point of using the AUI description as part of the development process. The preceding discussion shows that it is possible (and necessary for this work) to complete the shift to what Draheim et al. refer to as "the document-based GUI paradigm" [17]. Expanding the notion of the representation of the UI description to the realm of concurrent alternative representations, this can be extended as "the document-based UI paradigm". The advantages of this approach are significant, although there are also important trade-offs:

– *Separation of concerns between UI and application logic*
  This is an important technical requirement for AUI description languages [64], and the very use of AUI descriptions enforces this concept through the need for a well-defined mechanism to incorporate linking UI elements to program logic. This also implies a trade-off in flexibility, because the application logic is limited in its ability to directly interact with the UI.
– *Maintainability of the application*
  When a UI is described programmatically as part of the application, it typically will have a stronger dependency on system features such as the presentation toolkit that it is developed for. AUI descriptions do not exhibit this complication, because they are toolkit independent. In addition, the document-based nature of the AUI makes it much easier to modify the UI for small bug fixes, whereas a code-based UI requires changes in the application program code.
– *Adaptability*
  The adaptability of code-based UIs is generally limited to toolkit-level preference-controlled customisation. In contrast, the ability to make changes to the AUI description at runtime (e.g., by means of transformation rule sets) provides for a high level of adaptability.

The document-based UI paradigm is powerful, but it does impose some limitations on the designer/developer, because some very specialised toolkit features may not be available in all modalities. Toolkits for programmatically defined UI development generally offer a more rich feature set to the developer because they often allow access to the underlying lexical and syntactic elements. A rendering agent that creates a UI representation based on an AUI description offers a higher level of consistency and stability through the use of common higher level semantic constructs, at the cost of some flexibility.

While working towards universal access, it is important to be mindful of the creativity and æsthetic insight of designers. It is easy to reduce the user interface to its abstract semantic existence, but ultimately appearance does matter[30]. Empirical observation of both sighted and blind users as they operated computer systems has shown that there is a tendency to favour more æsthetically pleasing representations, and this seems to improve productivity. AUI descriptions in the PUIR framework therefore allow for rendering agent specific annotations to be added to the specification of UI elements[31]. The information is stored by the AUI engine for delivery to a rendering agent that requests it, but beyond that it is ignored by the AUI engine, because it is modality specific.

Despite the very powerful advantages of the document-based UI paradigm, it is important to recognise that the specification of the UI at the abstract semantic level does present a few complications, as illustrated by the following issues.

5.8.1 Dynamic user interfaces

It is common for UIs to contain elements that are not entirely statically defined, i.e., they contain information that is not known at development time. Prime examples are interaction objects that contain user modifiable data and elements that provide for user input. Another common occurrence is a UI element that only allows conditional interaction. GUIs often presents such elements as "grayed out", and they do not respond to user interaction while in that state. All these examples do not alter the composition of the UI presentation, and they therefore do not directly impact non-visual access.

A more disruptive feature involves truly dynamic updates in the user interface. A common example can be found in the almost iconic "File" menu on the application menu bar. It commonly displays (alongside various operations) a list of 5 or 10 most recently used files. The exact content and even the size of this list cannot be determined ahead of time. One possible solution may be the implementation of a feature in the AUI that specifies that this specific content must be queried from the application[32]. Alternatively, providing a facility for dynamic updates in the AUI description would provide a more generic solution to this type of problem.

Abstract user interfaces are commonly described in an XML-compliant UIDL, in a natural hierarchical structure, namely an object tree. Given that the PUIR framework renders the UI at runtime based on the AUI description, it is possible to support alterations to the user interface by means

[30] Although "appearance" is commonly interpreted as an aspect of visual perception, it actually carries a much broader meaning, across multiple modalities of perception.

[31] This is fundamentally different from other approaches (e.g., the HOMER UIMS [56,57]) where UI objects are described multiple times: once in abstract form, and once or more in modality-specific forms.

[32] This is commonly known as a "call back" feature.

of adding, removing, or updating parts of the hierarchy (subtrees). This ensures that dynamic UI changes are possible in a generic way. Components that render the actual representations can then receive a notification that an update is in order.

5.8.2 Legacy applications

The adoption of AUI-based application user interface design and development is still ongoing. It is therefore a reality that many legacy applications will not support a UI that is generated at runtime, based on an AUI description. Two possible approaches have been researched in recent years:

– *Reverse engineering the user interface*
  The UsiXML project includes techniques that make it possible to reverse engineering an existing (programmatically defined) UI, and obtain a representative AUI for the legacy application [6,41,66].
– *Interposing toolkit library implementations*
  This is essentially a form of reverse engineering by capturing all toolkit function calls using an API-compatible replacement library. It provides a non-invasive approach to capturing the programmatic construction of the UI. One such implementation was developed by the Visualisation and Interactive Systems Group at the University of Stuttgart [51].

5.8.3 "Creative programming"

By far the biggest obstacle towards providing non-visual access to GUIs is the occasional case of extreme use of features that are provided by user interface toolkits. In its worst form, an enthusiastic developer may implement his or her own toolkit, using a single large image widget from an existing toolkit as canvas for a custom rendering engine. Alternatively, an existing toolkit may be extended with some widgets that are not implemented in a compliant manner. Creative minds have even been known to implement buttons in dialog boxes that "run away" from the mouse pointer once it is within a predefined pixel-distance.

The only conclusion that can be reached in view of such creative programming is that it is not likely to be feasible to provide non-visual access to each and every application. It is obvious, however, that the use of techniques that result in this level of complexity are indicative of a sub-optimal design, because the separation between application logic and functionality, and visual presentation is lost.

5.9 AUI engine

As introduced at the beginning of this section (see page 17), the AUI engine is the core of the PUIR framework. It provides the following functionality:

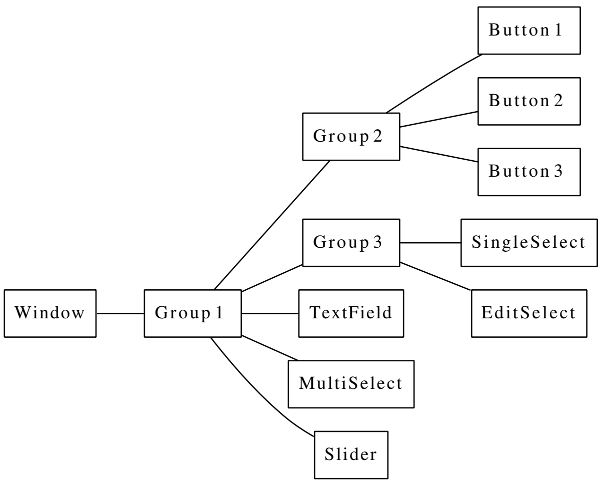


Fig. 19 Sample hierarchical AUI object model

– Translation of the AUI description in its textual UIDL form into the UI object model.
– Focus management (i.e., tracking which widget is to receive context-free input, such as keyboard input).
– Implementation of the user interaction semantics of UI elements.

The UI object model used in the AUI engine is a hierarchical model, using a tree structure to represent the UI. The root of the tree, the singleton node that does not have a parent, is the window. Children of the root node are by definition components in the UI. They have exactly one parent: a component that functions as a container, providing a way to group multiple components together in a logical and/or semantic unit. Components that are not containers appear as leaf nodes in the tree structure, whereas containers appear as internal nodes. Fig. 19 shows a partial UI object tree for a sample UI. In this figure, the parent-child relationship is represented by left-right connections, whereas top-down stacking represent grouping (sibling) relations. Note that this is similar to the commonly used off-screen model [35].

In order to maintain a strict differentiation between grouping and application semantics, a container can only function as a logical grouping of components. As such, it does not have any semantic user interaction associated with it. In fact, the only user interaction that is allowed for containers in the PUIR framework is related to establishing focus.

The objects in the AUI object model are abstract widgets (Table 3 list the supported widgets), and each was chosen specifically because of the fact that most (if not all) users are familiar with it. Note that the notion of using familiar concepts traces back to the initial design principles for the graphical user interface. Research has shown (e.g., Kurniawan, et al. [36,37], and Morley et al. [69,43]) that blind users have a good understanding of most UI elements in terms of their user interaction semantics. Being able to use the same familiar underlying concepts supports the use of

Table 3 Widgets provided by the AUI layer, by container

Widget	Description
window	Self-contained portion of a UI
Window	
group	Grouping of related widgets
menu bar	Container for menus
status bar	Notification message
tool bar	Container for easy-access widgets
Menu bar	
menu	Container for menu items
Menu (and item groups in a menu)	
menu	Sub-menu (contains menu items)
menu group	Grouping of menu items
menu item	Menu option that can be activated
mutex	Group of mutually exclusive toggles
toggle	Menu option that can be toggled
Mutex groups and menu mutex groups	
toggle	Mutually exclusive toggle
Group (non-menu)	
button	Button that can be activated
edit select list	Single-select list of items (provides a write-in option)
group	Logical grouping of widgets
multi select list	Multi-select list of items
mutex	Group of mutually exclusive toggles
single select list	Single-select list of items
text	Display text
textField	Text entry field
toggle	Selectable option
valuator	Ranged value entry

a single conceptual model, and offers stronger support for collaboration.

It is important to note that the abstract widgets listed in Table 3 are only "visible"[33] in terms of their semantics. Some widgets (as mentioned previously) serve as a container for parts of the UI, and they are therefore typically only noticeable by virtue of the effect they may have on focus traversal.

From the context of an application, the AUI engine will handle one or more windows simultaneously. In that sense, the application itself could be considered the root of the overall AUI object model. Yet, it is deliberately omitted in order to maintain separation of concerns.

5.9.1 Focus management

While conceptually the GUI is primarily based on the "seeing and pointing" design principle, empirical evidence shows that automated move-to-next-element functionality and keyboard navigation are essential components of productivity when text entry is required. The mental context switching between coordinating typing and pointer device movements seems to impose a delay[34].

Keyboard input (such as filling in text entry fields) operates without an implicit context as opposed to, e.g., pointer device operations. When the user operates a mouse in order to activate a button, the position of the pointer cursor determines what button is being activated. Keyboard input does not explicitly indicate what UI element it belongs to. Instead, an external focus management component takes care of this.

An element is said to be "in focus" or to "have focus" if it has been selected to receive user interaction events that are not explicitly associated with a UI element. The AUI engine manages the process of assigning focus to elements in three different ways:

– Programmatically: the application can request that focus be moved to the next or the previous UI element in the focus traversal order (see below for more information). It can also request focus to be given to a specific element. Aside from the application, it may also be beneficial to allow widgets to do the same, e.g., as a default action after an operation is completed.
– User interaction event: the user can navigate the window by moving between UI elements by means of direct user interaction. This is most often used for keyboard navigation (and exploration) based on the focus traversal order. Common navigation operations are: move to next element, move to previous element, move to next group, move to previous group, ...
– User interaction side-effect: when the user performs an operation on a UI element that is not currently in focus, it is common practice to shift focus to that element. This is essentially a special case of the programmatical assignment of focus.

The "focus traversal order" mentioned in the list above refers to a well defined strict ordering of elements across the hierarchy of objects in the AUI object model. It specifies in what order the various elements receive focus, in a strict linear order. Containers are not included in the list because they have no semantic user interaction associated with them.

The AUI engine defines the order in the AUI object model tree as depth-first, left to right. Based on the example in Fig. 19 (where the order is rightmost-first, top to bottom because the tree is flipped horizontally for display purposes), the focus traversal order will be: *Button 1*, *Button 2*, *Button 3*, *ComboBox*, *EditComboBox*, *TextField*, *List*, *Slider*.

[33] In this context, being "visible" means that the user can note the existence of the widget. Being part of the AUI, the widget obviously has no perceptual characteristics.

[34] More specific research into the impact of mental context switching and related topics is outside the scope of this work.

5.9.2 User interaction semantics

As mentioned in section 5.6, the AUI engine is responsible for providing an implementation of the user interaction semantics of all widgets. This is crucial in the design of the PUIR framework, because it ensures that the behaviour of UI elements is independent from the representation of the UI.

Various user interaction operations are supported by the PUIR design:

- *Action*: This interaction is used to trigger a specific operation or function. It is one of the most basic forms of the “Seeing and Pointing” design principle for GUIs, because it captures the familiar action of pressing the “On/Off” button on an appliance.
- *Container*: All objects in the UI are encapsulated in a container. Whenever its content changes (adding an object or removing one), an operation of this type takes place. Creating a new object results in adding the object to a container, whereas deleting an object causes it to be removed from its container. The well-known “Drag and Drop” GUI interaction can be interpreted as a combinationc of removing an object from one container and adding it to another.
- *Focus*: This is probably the most obscure of all forms of user interaction. It captures the notion of what the user’s attention is focused on. When a person is filling out a form on paper, it is common to visually locate a specific item, and to then bring one’s pen to the writing space that is associated with that item. The person truly remains focused on the item he or she is filling out.
- *Selection*: When a user is presented with multiple options with the restriction that only one can be chosen, a selection process takes place. Often, a user will consider several options (selecting an option, only to then later dismiss that selection), until a final choice is made, which is then finalised (by means of an *Action* operation).
- *SetSelection*: It is sometimes appropriate to select more than one item from a list of choices (e.g., a buffet). Items that are part of the selection may be consecutive or separate. It is also quite likely that the selection set may change while the user makes up his or her mind. When finally a decision is reached, the appropriate selection is finalised (by means of an *Action* operation).
- *TextCaret*: This operation is (alike the Focus operation) quite obscure because it also relates to the location on which the user is focusing his or her attention. This operation targets text, and is used as the equivalent of placing one’s pen tip in a specific position (at a particular character in the text string).
- *TextSelection*: When a user intends to select some text, he or she will commonly utilise a method that requires the least effort. If the user knows that an entire line of text is to be selected, locating any point on that line is generally sufficient. Likewise, when trying to select a specific word, any point within the word boundaries is usually acceptable. Only when the text selection is more complex, will a user specifically select starting and end points by character.
- *ValueChange*: Any element that represents a dynamic value, i.e., an element for which the user can select or input a specific value, supports user interaction that modifies the current value. The changing of the value is not a final operation, as it is not uncommon for a user to change their mind (even multiple times) before deciding on the final value (which is then finalised by means of an *Action* operation).
- *Visibility*: For widgets that are not always represented in the UI, a conceptual characteristic of visibility can be assigned. While ordering food in a restaurant, a person typically consults a menu. Once the desired selection is made and communicated to the waiter, the menu is often taken away by the waiter, or it is put aside. No one pays any attention to it unless a followup order (or the intent to) is anticipated.

Table 4 Operations supported by abstract widgets

Containers	**Operation(s)**
group	Container
menu	Container, Focus, Visibility
menu bar	Container
mutex	Container
status bar	Container, ValueChange
tool bar	Container
window	Container, Focus, Visibility
Components	**Operation(s)**
button	Focus, Action
edit select list	Focus, Selection, ValueChange, Action
multi select list	Focus, SetSelection, Action
menu item	Focus, Action
single select list	Focus, Selection, Action
text	ValueChange
text field	Focus, TextCaret, TextSelection, ValueChange, Action
toggle	Focus, Action
valuator	Focus, ValueChange, Action

User interaction is presented to the AUI engine as semantic events, targeted at the widget it operates on. The operations that each widget supports are listed in Table 4. The abstract widget implementation handles the event, and (usually) dispatches notification events to rendering agents and the application to indicate that the semantic operation has been processed.

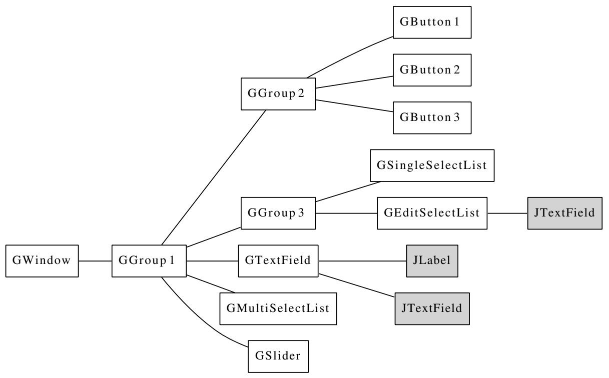


Fig. 20 Sample hierarchical CUI object model for the AUI in Fig. 19 *The clear boxes indicate components that are created in one-to-one correspondence with the AUI model. The shaded components are necessary additions in order to render the UI correctly in the specific rendering agent.*

5.10 Rendering agents

The AUI engine described in section 5.9 operates entirely within the context of the abstract UI object model, at the conceptual level. In order to be able to present the user with a UI representation within the context of a specific modality, the AUI must go through a reification process. This part of the PUIR framework operates at the perceptual level and is provided by the rendering agents.

Each rendering agent provides AUI reification within the context of one or more modalities. This is generally done as a two-step process:

1. On request of the AUI engine, a concrete UI object model is constructed, incorporating a specific "Look & Feel" based on an established set of interaction metaphors.
2. Based on a modality-specific presentation toolkit, the CUI from the previous step is finalised into the FUI that is presented to the user.

5.10.1 Mapping the AUI model onto a CUI model

Fig. 20 provides a (partial) example of the CUI object model that a rendering agent might build based on the AUI object model that resides with the AUI engine (Fig. 19). As illustrated in this example, there is no guarantee for a one-to-one correspondence between the two models, because abstract widgets may very well map onto multiple concrete widgets. This is the case for the abstract TextField widget that, e.g., as part of a visual representation agent based on Java Swing is presented as a JLabel object and a JTextField object.

The reverse is certainly possible as well. A rendering agent may have no need for some intermediary container objects, and thereby map multiple abstract widgets onto a single more complex presentation widget. Note that regardless of the mappings between models, the user interaction semantics remain the same.

5.10.2 User interaction

In support of the separation of concerns concept as suggested by Parnas [49], there is no direct communication between the application and the rendering agents whatsoever. Any and all requests (be it from the application or the rendering agent) are to be processed by the AUI engine, which will then provide notification to the rendering agents. Upon receiving a notification event, a determination is made whether the operation requires rendering the change in the representation of the UI.

Many commonly available presentation toolkits provide an implementation for both the presentation and interaction components of the UI, rendering the effects of a user interaction immediately, and providing a notification or call back mechanism to the application to allow the program logic to react to the user interaction event. Within the requirements of the PUIR design principles, if a presentation toolkit contains a user interaction component, any events from this component must be forwarded to the AUI engine for processing, and the presentation of UI changes as a result of the interaction (e.g., visually showing that a button was pressed, or providing auditory feedback for the same action) must only take place as a response to receiving a notification event from the AUI engine that a specific semantic operation took place. Without this clear separation it would be very difficult to ensure coherence between parallel representations[35].

It is important to note that rendering agents may implement context-specific user interaction. This level of interaction is independent from the actual UI and the application semantics, and therefore not restricted to processing by the AUI engine. Being able to provide this level of interaction is important in order to support exploration of the UI in alternative representations, as a solution to one of the HCI issues related to the usability of alternative UIs (see section 3.3). The user interaction provided by the rendering agent must not result in actual UI interaction, and it therefore operates as a distinct UI mode, i.e., all user input is interpreted as exploration-only operations.

5.10.3 Queries to the AUI engine

The rendering agent must also be able to query information from the AUI engine, as needed. Such queries are almost always in response to receiving an event from the AUI engine, but there may be legitimate reasons for the rendering agent to spontaneously request some data from the AUI engine. The most common use is to request additional information

[35] A common problem would be that the modality in which the user interaction was initiated might render the feedback prior to the application logic responding to the operation, whereas all other renderings would render feedback afterwards. This is also commonly observed in assistive technology solutions such as screen readers that are implemented as a derivative to the graphical representation.

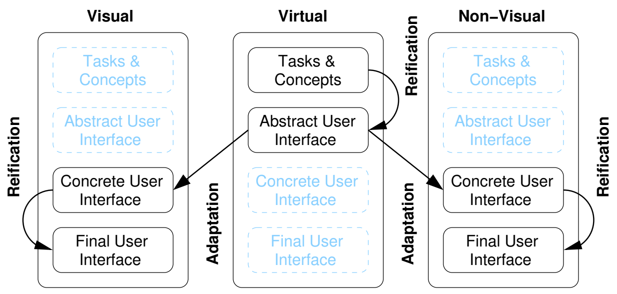


Fig. 21 URF diagram for PUIR

about a component in the AUI object model in order to render that component.

5.10.4 Modality-specific limitations

The rendering agent may impose some limitations on the overall UI due to modality-specific limitations. Due to the significant impact imposed on the entire UI, care must be taken to only require this when absolutely necessary. One common relatively low-impact limiting requirement is synchronisation. When only a single representation is used, the UI is primarily self-synchronising because the user cannot interact with components that are not rendered yet. In the presence of parallel representations, it may be necessary to ensure that all agents are presenting the same state of the UI at any given time. Latency with some modalities may therefore require delays to be inserted into the flow of interaction.

5.11 Analysis

In the context of the Unified Reference Framework presented in section 3.1, the Parallel User Interface Rendering approach can be described schematically as shown in Fig. 21.

In comparison with the URF diagram for the Fruit system (Fig. 5) and HOMER UIMS (Fig. 8), it is clear that from a model perspective PUIR is a combination of Fruit and HOMER UIMS. One might say that it is the best of both worlds.

The user interface for an application is designed as a reification process from a tasks and concepts definition to an abstract UI, independent from any modality. Rendering in specific modalities is accomplished as a cross-level adaptation operation (adaptation combined with reification), constructing a concrete user interface based on the AUI. Further reification yields the final UI. Note that the adaptation step from AUI to CUI can be performed for multiple contexts of use (modalities) simultaneously.

In this model, all interaction between the application and the UI is handled by the AUI, whereas interaction between a user and the system is the responsibility of the rendering agents.

In consideration of the non-visual access concerns expressed in section 3.3, it is obvious that static and dynamic coherence can be guaranteed by the PUIR approach, given that all representations are derived from the same source. By careful implementation of the reification and adaptation operations in UI representation construction, it is possible to ensure the Equivalence CARE property. Likewise, conveying meaningful graphical information is assured because the rendering agents receive all events, be they related to visual aspects or not.

Input is processed at the level of the rendering agent, and translated into semantic events prior to being passed on to the AUI engine. This allows the rendering agents to implement modality-specific (or need-specific) exploration features that are guaranteed to not interfere with the operation of the system. It also provides the freedom to provide user interaction operations that are tailored to the needs of the user.

As an approach to providing equivalent representations of multi-modal user interfaces, and in view of the fact that a single conceptual model lies at the basis of this novel approach, the short analysis presented in this section demonstrates that this technique satisfies all requirements.

6 Conclusions and Future Work

Parallel User Interface Rendering is proving to be a very powerful technique in support of the Design-for-All and Universal Access principles. It builds on a solid base of research and development of abstract user interfaces and UIDLs, and it provides a framework where non-visual rendering of the UI operates at the same level as the visual rendering rather than as a derivative. The design principles build a foundation for a very powerful approach. Especially user collaboration benefits greatly from this approach because of the coherence between all renderings, allowing communication about user interaction to be based on a substantially similar mental model of the user interface.

The current proof-of-concept implementation is based on a custom UIDL, but for further development[36] collaboration with a UIDL is expected. The decision to continue with a broader interpretation of the original GUI design principles drives the choice of underlying UIDL. The ability of, e.g., UsiXML to capture the UI with models at different levels of abstraction can be a real asset to this novel approach.

It important to recognise that any approach that allows for a high degree of flexibility increases the risk that someone will use that flexibility in a way that interferes with the very design principles that the presented solution is based upon. This work does not intend to guarantee that any UI

[36] And in order to work towards a possible future adoption as an AT support solution.

developed within the PUIR framework will be 100% accessible. However, the presented work promotes sound UI design, and ensures that at a minimum each user is aware of the existence of UI elements, even if it is possible that in rare cases 100% functional user interaction cannot be assured due to potential improper use of features.

The PUIR framework can contribute to the field of accessibility well beyond the immediate goal of providing nonvisual access to GUIs. The generic approach behind the PUIR design lends itself well to developing alternative rendering agents in support of other disability groups. Because rendering agents need not necessarily execute local to applications, accessible remote access is possible as well. The PUIR framework may also benefit automated application testing, by providing a means to interact with the application programmatically without any dependency on a specific UI rendering.

Acknowledgements The research presented in this paper is part of the author's doctoral work at the Katholieke Universiteit Leuven, Belgium, under supervision by Jan Engelen (ESAT-SCD-Research Group on Document Architectures).

References

1. Ali, M.F.: A transformation-based approach to building multiplatform user interfaces using a task model and the user interface markup language. Ph.D. thesis, Virginia Polytechnic Institute and State University (2004)
2. Barnicle, K.: Usability testing with screen reading technology in a Windows environment. In: Proceedings of the 2000 Conference on Universal Usability, CUU '00, pp. 102–109. ACM (2000)
3. Bergman, E., Johnson, E.: Toward accessible human-computer interaction. In: J. Nielsen (ed.) Advances in human-computer interaction (vol. 5), pp. 87–113. Ablex Publishing Corp. (1995)
4. Bishop, J., Horspool, N.: Developing principles of GUI programming using views. In: Proceedings of the 35th SIGCSE technical symposium on Computer science education, SIGCSE '04, pp. 373–377. ACM (2004)
5. Blattner, M., Glinert, E., Jorge, J., Ormsby, G.: Metawidgets: towards a theory of multimodal interface design. In: Computer Software and Applications Conference, 1992. COMPSAC '92. Proceedings., Sixteenth Annual International, pp. 115–120. IEEE Computer Society Press (1992)
6. Bouillon, L., Vanderdonckt, J., Chow, K.C.: Flexible re-engineering of web sites. In: Proceedings of the 9th international conference on Intelligent user interfaces, IUI '04, pp. 132–139. ACM (2004)
7. Bouraoui, A., Soufi, M.: Improving computer access for blind users. In: K. Elleithy (ed.) Advances and Innovations in Systems, Computing Sciences and Software Engineering, pp. 29–34. Springer Netherlands (2007)
8. Braille Authority of North America: Braille Formats: Principles of Print to Braille Transcription 1997. American Printing House for the Blind (1998)
9. Brunet, P., Feigenbaum, B.A., Harris, K., Laws, C., Schwerdtfeger, R., Weiss, L.: Accessibility requirements for systems design to accommodate users with vision impairments. IBM Syst. J. **44**(3), 445–466 (2005)
10. Buxton, W., Gaver, W., Bly, S.: Auditory interfaces: The use of non-speech audio at the interface (1994). Draft manuscript
11. Calvary, G., Coutaz, J., Thevenin, D.: A unifying reference framework for the development of plastic user interfaces. In: Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction, EHCI '01, pp. 173–192. Springer-Verlag (2001)
12. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with Computers **15**(3), 289–308 (2003)
13. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Florins, M., Vanderdonckt, J.: Plasticity of user interfaces: A revised reference framework. In: Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design, pp. 127–134. INFOREC Publishing House Bucharest (2002)
14. Chapman, M.: Create user interfaces with glade. Linux J. **2001**(87), 90–92,94 (2001)
15. Congress of the United States of America: 42 U.S.C. – The Public Health and Welfare, Section 1382(a)2). GPO (1997)
16. Coutaz, J., Nigay, L., Salber, D.: Multimodality from the user and system perspectives. In: Proceedings of the ERCIM'95 workshop on Multimedia Multimodal User Interfaces (1995)
17. Draheim, D., Lutteroth, C., Weber, G.: Graphical user interfaces as documents. In: Proceedings of the 7th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: design centered HCI, CHINZ '06, pp. 67–74. ACM (2006)
18. Edwards, A.D.N.: The difference between a blind computer user and a sighted one is that the blind one cannot see (1994). Interactionally Rich Systems Network, Working Paper No. ISS/WP2
19. Edwards, A.D.N., Mitsopoulos, E.: A principled methodology for the specification and design of nonvisual widgets. ACM Trans. Appl. Percept. **2**(4), 442–449 (2005)
20. Edwards, W.K., Mynatt, E.D., Stockton, K.: Providing access to graphical user interfaces – not graphical screens. In: Proceedings of the first annual ACM conference on Assistive technologies, Assets '94, pp. 47–54. ACM (1994)
21. Gajos, K., Weld, D.S.: SUPPLE: automatically generating user interfaces. In: IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces, pp. 93–100. ACM Press (2004)
22. Gaver, W.W.: The sonicfinder: an interface that uses auditory icons. Hum.-Comput. Interact. **4**(1), 67–94 (1989)
23. Gunzenhäuser, R., Weber, G.: Graphical user interfaces for blind people. In: K. Brunnstein, E. Raubold (eds.) 13th World Computer Congress 94, Volume 2, pp. 450–457. Elsevier Science B.V. (1994)
24. Haneman, B., Mulcahy, M.: The GNOME accessibility architecture in detail (2002). Presented at the CSUN Conference on Technology and Disabilities
25. Harness, S., Pugh, K., Sherkat, N., Whitrow, R.: Fast icon and character recognition for universal access to WIMP interfaces for the blind and partially sighted. In: E. Ballabio, I. Placencia-Porrero, R.P.d.l. Bellcasa (eds.) Rehabilitation Technology: Strategies for the European Union (Proceedings of the First Tide Congress), pp. 19–23. IOS Press, Brussels (1993)
26. Hollins, M.: Understanding Blindness: An Integrative Approach. Lawrence Erlbaum Associates (1989)
27. Institute of Electrical and Electronics Engineers: 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology. IEEE, Los Alamos, CA (1990)
28. International Organization for Standardization: ISO/IEC 9126, Information Technology, Software Product Evaluation, Quality Characteristics and Guidelines for their Use. ISO, Geneva (1991)
29. International Organization for Standardization: ISO/IEC 9241-11, Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs), Part 11: Guidance on Usability. ISO, Geneva (1998)

30. Jacob, R.J.K.: User interfaces. In: A. Ralston, E.D. Reilly, D. Hemmendinger (eds.) Encyclopedia of Computer Science, Fourth Edition. Grove Dictionaries, Inc. (2000)
31. Kawai, S., Aida, H., Saito, T.: Designing interface toolkit with dynamic selectable modality. In: Proceedings of the second annual ACM conference on Assistive technologies, Assets '96, pp. 72–79. ACM (1996)
32. Kay, A.C.: User interface: A personal view. In: B. Laurel (ed.) The Art of Human-Computer Interface Design, pp. 191–207. Addison-Wesley Publishing Co. (1990)
33. Kochanek, D.: Designing an offscreen model for a gui. In: W. Zagler, G. Busby, R. Wagner (eds.) Computers for Handicapped Persons, *Lecture Notes in Computer Science*, vol. 860, pp. 89–95. Springer Berlin / Heidelberg (1994)
34. Kost, S.: Dynamically generated multi-modal application interfaces. Ph.D. thesis, Technische Universität Dresden, Dresden, Germany (2006)
35. Kraus, M., Völkel, T., Weber, G.: An off-screen model for tactile graphical user interfaces. In: K. Miesenberger, J. Klaus, W. Zagler, A. Karshmer (eds.) Computers Helping People with Special Needs, *Lecture Notes in Computer Science*, vol. 5105, pp. 865–872. Springer Berlin / Heidelberg (2008)
36. Kurniawan, S.H., Sutcliffe, A.G.: Mental models of blind users in the Windows environment. In: K. Miesenberger, J. Klaus, W. Zagler (eds.) Computers Helping People with Special Needs, *Lecture Notes in Computer Science*, vol. 2398, pp. 373–386. Springer Berlin / Heidelberg (2002)
37. Kurniawan, S.H., Sutcliffe, A.G., Blenkhorn, P.L.: How blind users' mental models affect their perceived usability of an unfamiliar screen reader. In: M. Rauterberg, M. Menozzi, J. Wesson (eds.) Human-Computer Interaction INTERACT '03, pp. 631–638. IOS Press (2003)
38. Laberge-Nadeau, C.: Wireless telephones and the risk of road crashes. Accident Analysis & Prevention **35**(5), 649–660 (2003)
39. Lauridsen, O.: Abstract specification of user interfaces. In: Conference companion on Human factors in computing systems, CHI '95, pp. 147–148. ACM (1995)
40. Lévesque, V.: Blindness, technology and haptics. Tech. Rep. TR-CIM-05.08, McGill University, Centre for Intelligent Machines, Haptics Laboratory (2008)
41. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., Trevisan, D.: UsiXML: A user interface description language for context-sensitive user interfaces. In: K. Luyten, M. Abrams, J. Vanderdonckt, Q. Limbourg (eds.) Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages", pp. 55–62 (2004)
42. de Melo, G., Honold, F., Weber, M., Poguntke, M., Berton, A.: Towards a flexible ui model for automotive human-machine interaction. In: Proceedings of the 1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications, AutomotiveUI '09, pp. 47–50. ACM (2009)
43. Morley, S.: Window Concepts: An Introductory Guide for Visually Disabled Users. Royal National Institute for the Blind (1995)
44. Mynatt, E.D.: Transforming graphical interfaces into auditory interfaces for blind users. Hum.-Comput. Interact. **12**(1), 7–45 (1997)
45. Mynatt, E.D., Edwards, W.K.: Mapping guis to auditory interfaces. In: Proceedings of the 5th annual ACM symposium on User interface software and technology, UIST '92, pp. 61–70. ACM (1992)
46. Mynatt, E.D., Weber, G.: Nonvisual presentation of graphical user interfaces: contrasting two approaches. In: Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence, CHI '94, pp. 166–172. ACM (1994)
47. Newell, A.F.: CHI for everyone. Interfaces **35**, 4–5 (1997)
48. Nigay, L., Coutaz, J.: A design space for multimodal systems: concurrent processing and data fusion. In: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems, CHI '93, pp. 172–178. ACM (1993)
49. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Commun. ACM **15**(12), 1053–1058 (1972)
50. Pontelli, E., Gillan, D., Xiong, W., Saad, E., Gupta, G., Karshmer, A.I.: Navigation of HTML tables, frames, and XML fragments. In: Proceedings of the fifth international ACM conference on Assistive technologies, Assets '02, pp. 25–32. ACM (2002)
51. Rose, D., Stegmaier, S., Reina, G., Weiskopf, D., Ertl, T.: Non-invasive adaptation of black-box user interfaces. In: Proceedings of the Fourth Australasian user interface conference on User interfaces 2003 - Volume 18, AUIC '03, pp. 19–24. Australian Computer Society, Inc. (2003)
52. Sacks, O.: The mind's eye: What the blind see. The New Yorker pp. 48–59 (2003)
53. Sadato, N., Pascual-Leone, A., Grafman, J., Deiber, M.P., Ibañez, V., Hallett, M.: Neural networks for braille reading by the blind. Brain **121**, 1213–1229 (1998)
54. Sadato, N., Pascual-Leone, A., Grafman, J., Ibañez, V., Deiber, M.P., Dold, G., Hallett, M.: Activation of the primary visual cortex by braille reading in blind subjects. Nature **380**(6574), 526–528 (1996)
55. Savidis, A., Stephanidis, C.: Building non-visual interaction through the development of the rooms metaphor. In: Conference companion on Human factors in computing systems, CHI '95, pp. 244–245. ACM (1995)
56. Savidis, A., Stephanidis, C.: Developing dual user interfaces for integrating blind and sighted users: the HOMER UIMS. In: Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '95, pp. 106–113. ACM Press/Addison-Wesley Publishing Co. (1995)
57. Savidis, A., Stephanidis, C.: The HOMER UIMS for dual user interface development: Fusing visual and non-visual interactions. Interacting with Computers **11**(2), 173–209 (1998)
58. Seybold, J.: Xerox's "star". The Seybold Report **10**(16) (1981)
59. Smith, D.C., Harslem, E.F., Irby, C.H., Kimball, E.B., Verplank, W.L.: Designing the Star User Interface. BYTE pp. 242–282 (1982)
60. Souchon, N., Vanderdonckt, J.: A review of XML-compliant user interface description languages. In: J.A. Jorge, N. Jardim Nunes, J. Falcão e Cunha (eds.) Interactive Systems. Design, Specification, and Verification, *Lecture Notes in Computer Science*, vol. 2844, pp. 391–401. Springer Berlin / Heidelberg (2003)
61. Stephanidis, C., Savidis, A.: Universal access in the information society: Methods, tools, and interaction technologies. Universal Access in the Information Society **1**(1), 40–55 (2001)
62. Sun Microsystems: GNOME 2.0 desktop: Developing with the accessibility framework. Tech. rep., Sun Microsystems (2003)
63. Theofanos, M.F., Redish, J.G.: Bridging the gap: between accessibility and usability. interactions **10**(6), 36–51 (2003)
64. Trewin, S., Zimmermann, G., Vanderheiden, G.: Abstract user interface representations: how well do they support universal access? In: Proceedings of the 2003 conference on Universal usability, CUU '03, pp. 77–84. ACM (2003)
65. Trewin, S., Zimmermann, G., Vanderheiden, G.: Abstract representations as a basis for usable user interfaces. Interacting with Computers **16**(3), 477–506 (2004)
66. Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M.: UsiXML: a user interface description language for specifying multimodal user interfaces. In: WMI '04: Proceedings of the W3C Workshop on Multimodal Interaction (2004)
67. Weber, G.: Programming for usability in nonvisual user interfaces. In: Proceedings of the third international ACM conference on Assistive technologies, Assets '98, pp. 46–48. ACM (1998)

68. Weber, G., Mager, R.: Non-visual user interfaces for X Windows. In: Proceedings of the 5th international conference on Computers helping people with special needs. Part II, pp. 459–468. R. Oldenbourg Verlag GmbH (1996)
69. Weber, G., Petrie, H., Kochanek, D., Morley, S.: Training blind people in the use of graphical user interfaces. In: W. Zagler, G. Busby, R. Wagner (eds.) Computers for Handicapped Persons, *Lecture Notes in Computer Science*, vol. 860, pp. 25–31. Springer Berlin / Heidelberg (1994)