

PUIR: Parallel User Interface Rendering

Kris Van Hees and Jan Engelen

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

Abstract. While providing non-visual access to graphical user interfaces has been the topic of research for over 20 years, blind users still face many obstacles when using computer systems. The higher degree of flexibility for both developers and users poses additional challenges. Existing solutions are largely based on either graphical toolkit hooks, queries to the application and environment, scripting, or 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. PUIR provides the mechanism to render a user interface simultaneously in multiple forms (e.g. visual and non-visual).

1 Introduction

Over the past few years, our world has become more and more infused with devices that feature graphical user interfaces (GUIs), ranging from home appliances with LCD displays to mobile phones with touch screens and voice control. The emergence of GUIs poses a complication for blind users due to the implied visual interaction model. Even in the more specific field of general purpose computer systems at home and in the work place, non-visual access often still poses a problem, especially in Unix-based environments where mixing graphical visualisation toolkits is very common. While popularity keeps growing for this group of systems, advances in accessibility technology in support of blind users remain quite limited.

Currently available solutions use a combination of toolkit extensions, scripting, and complex heuristics to obtain sufficient information to make non-visual renderings possible [1,2]. Alternative approaches use model-driven user interface (UI) composition at development time, or runtime adaptation [3,4]. Leveraging the adoption of UI development using abstract user interface (AUI) definitions [5,6], it is possible to build a solid base for providing non-visual access as an integral component of the UI environment [7,8]. Rather than trying to interpret a visual presentation in a non-visual context, the renderings are presented in parallel from the same abstract UI description. This also ensures that information

on semantic relationships between UI elements is available across all presentations, eliminating problems such as trying to guess what “label” belongs to an input field in the UI.

The Parallel User Interface Rendering (PUIR) technique expands upon UI development by means of AUIs by providing a framework in which multiple rendering agents can provide presentations of the UI in parallel [9]. The application can remain completely unaware of the specific renderings that are presented to the user.

While PUIR is rendering-independent, simultaneous rendering in visual and non-visual modalities has been selected as the proof-of-concept. This choice is in line with the overall scope of the research that led to the PUIR design, and builds on extensive past research on challenges that blind users face when operating a computer system.

The remainder of this paper first presents related work on GUI accessibility and UI abstraction. The third section provides a description of the PUIR framework. This is followed by more in-depth implementation details, and the conclusion, offering a description of future work.

2 Related Work

Accessibility of GUIs for blind users has been the topic of research for many years. Mynatt and Weber discussed two early approaches [10], 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 [11].

Savidis and Stephanidis researched alternative interaction metaphors for non-visual user interfaces [12], which formed the basis for the HAWK toolkit [13]. It provides interaction objects and techniques that have been designed specifically for non-visual access. The AVANTI project [4] uses this toolkit in its context-based runtime user interface adaptation mechanism. Gajos and Weld also propose a UI adaptation mechanism [3], approaching the problem of different user and environment models as a search for the most optimal UI representation given specific device characteristics and a user interaction model.

The Visualisation and Interactive Systems Group of the University of Stuttgart describes a system for black-box UIs [14], introducing the concept of replacing a user interface by interposing libraries. It shows the power of capturing UI information at the application-toolkit crossover rather than obtaining similar information inside the toolkit implementation, even though the described system of non-invasive adaptation still depends on programmatic UI construction.

The use of AUIs in UI development lies at the core of the UsiXML [5] project at the Belgian Laboratory of Computer-Human Interaction (BCHI) at the Université Catholique de Louvain. Earlier work on application UIs and Word Wide Web forms provides important insights into the obstacles that blind users face when dealing with user interaction model, as reported by Barnicle [15], Pontelli et al [16], and Theofanos/Redish [17].

The concept of parallel user interface rendering as a way to provide non-visual access to GUIs emerged from doctoral research into the use of abstract UI definitions as basis for accessibility technologies [7,8,9]. Stefan Kost's GiTK project [6] also builds on the value of UI abstraction to provide dynamically generated user interfaces. His work briefly touches on the topic of multiple interfaces, offering some ideas for future work in this area.

3 Parallel UI Rendering

The PUIR framework is shown schematically in Fig. 1. It provides for multiple simultaneous runtime interpretations of a given abstract UI description. Rather than constructing the UI by means of function calls into a specific widget toolkit, application developers define the UI in an abstract form, often by means of design tools. The result is an AUI description in a standardised form (e.g. UsiXML [5], GIML [6], ...). This description is interpreted by the AUI engine.

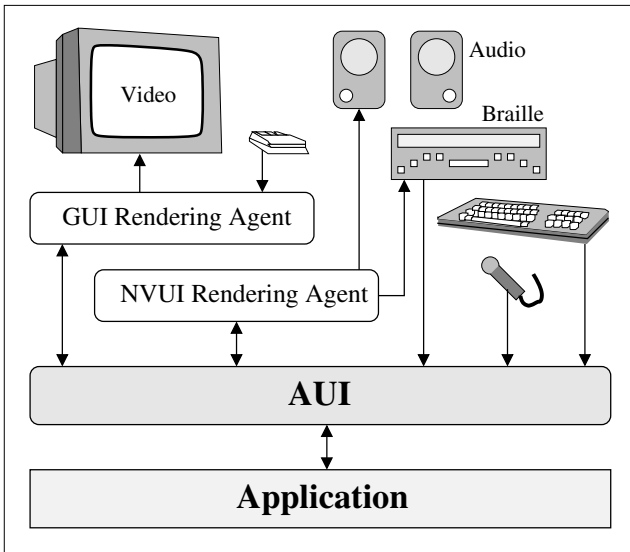


Fig. 1. Schematic overview of PUIR

The presentation of the UI is delegated to one or more rendering agents, each providing its own interpretation of the UI in one or more modalities. This technique does require the AUI to be able to represent both data and a description of how to present that data, as suggested by Trewin [18]. It also promotes the AUI handling to active component status in order to support rendering-independent semantics for the UI. This also requires the AUI engine to not depend on any implementation details of the rendering agents.

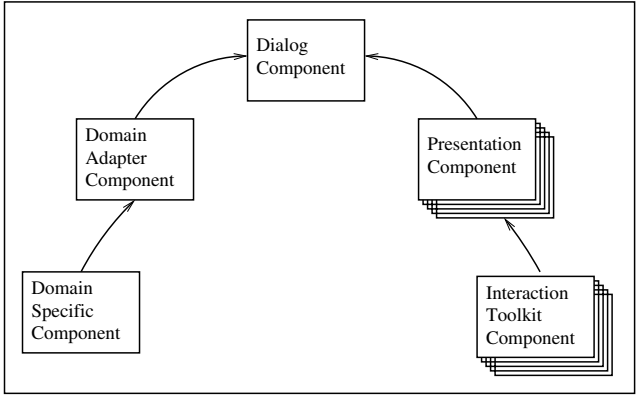


Fig. 2. ARCH model adapted for PUIR

Various models for user interface management systems (UIMS) have been suggested in past research. One model with a reasonably high degree of modularisation is the ARCH model [19]. The PUIR framework maps well onto this model, if we account for the novel features that parallel rendering introduces as shown in Fig. 2. The following five components are identified:

- **Domain Specific Component:** This component is provided by the application itself. The PUIR framework does not cover this component, other than providing a mechanism for the application to interact with the UI and to retrieve information from the UI.
- **Domain Adaptor Component:** The abstract UI description covers this component. It defines the UI from the perspective of the application, and it serves as the source for the AUI engine. The UI element descriptions can contain rendering agent specific annotations that control cosmetic aspects of the element. These annotations are ignored by the AUI engine.
- **Dialog Component:** This function is provided by the AUI engine, an active component in the PUIR framework. It implements the interaction semantics of UI elements, it encapsulates the UI data elements, it provides synchronised event processing for all semantic UI events, and it implements UI focus management.
- **Presentation Component:** Rendering agents serve as presentation components. The ARCH model does not address the possibility of multiple instances of a specific component. Within the PUIR framework, the collection of implemented rendering agents constitutes the presentation component of the ARCH model.
- **Interaction Toolkit Component:** Each rendering agent utilises a toolkit that provides functionality to present the UI elements in one or more output modalities. In terms of the ARCH model, every instance of the *Presentation Component* is associated with an *Interaction Toolkit Component*.

The parallel rendering that PUIR provides imposes important requirements on the UIMS design. Not only does it require a well defined application programming interface (API) between the AUI engine and the rendering engine, it also requires a modified way to handle input device processing.

The AUI engine implements the user interaction semantics of the UI elements, their interactions, and input focus (i.e. what UI element receives textual input). Some input devices relate directly to one or more modalities handled by a specific rendering agent while others are more independent:

- Mouse operations are inherently graphical in nature, operating in the two dimensional graphical space in which the UI is rendered. Positional information needs to be processed by the rendering agent, and where needed semantic events will be dispatched to the AUI engine.
- Braille keyboard input usually combines coded input (Braille cells) with a limited form of cursor addressable input (positional input). The coded input gets translated into textual input, and can then be handled as regular keyboard input.
- Keyboard input is not directly related to any specific rendering agent. It gets interpreted in terms of focus on a specific UI element.

4 Implementation

A proof-of-concept implementation of the PUIR framework has been implemented in Java, as a generic UI toolkit. It provides 25 UI elements that are commonly used for applications, such as buttons, text input fields, labels, editable comboboxes, mutually exclusive toggle lists, ... The rest of this section describes important implementation details.

4.1 The AUI Engine

The AUI engine interprets an abstract UI description provided in an XML file, and constructs the user interface. The application can operate on its UI by making direct calls to elements at the AUI level. In return, the AUI engine dispatches events to the application to provide notification of various state changes. The AUI engine also implements focus management.

Interactions between the AUI engine and rendering agents are also handled through event dispatching, whereas direct calls from rendering agent objects to AUI engine objects are used to query specific information. This design supports the requirement that the AUI engine has no knowledge about the implementation details of the rendering agents.

4.2 Rendering Agents

The PUIR framework requires that rendering agents implement a specific API. The experimental implementation provides for in-process rendering agents and remote rendering agents. Events dispatched by the AUI engine are sent to all rendering agents for processing, to ensure consistency between the presentations.

In-Process Rendering Agents. The proof-of-concept implementation provides a Java Swing based in-process rendering agent. Due to the design of widgets in the Swing toolkit, the rendering agent turned out to be quite complex. The PUIR framework requires user interaction semantics to be handled in the AUI engine, whereas Java Swing widgets implement semantics within the toolkit. Various aspects of widget functionality have been bypassed to ensure that any and all interaction with the UI elements is triggered by events dispatched from the AUI engine.

The swing rendering agent provides a graphical presentation of the UI, and implements an interaction model for mouse operations. While most mouse events are ignored because they have no semantic significance, select events are translated into their PUIR equivalents and dispatched to the AUI engine.

Remote Rendering Agents. A speech output presentation of the UI is provided as a remote rendering agent, communicating with applications by means of a message bus. Regardless of the number of remote rendering agents, only a single stub is instantiated within the application. The message bus handles multiplexing between applications and remote rendering agents. The custom communication protocol provides for mutual discovery, event passing, and request/response dialogues.

The actual rendering agent operates on a proxy of the actual UI, providing both transparency and caching.

4.3 User Input

The proof-of-concept implementation supports three input modalities:

- Keyboard input: this is processed directly at the AUI engine level. While this is a simplification of the design, it does not actually impact the expected functionality.
- Mouse input: as described above, mouse input is processed by the in-process graphical rendering agent. Relevant events are presented to the AUI engine for processing.
- Braille keyboard input: Braille input (albeit in coded form) is transcribed into equivalent keyboard input, and presented to the AUI engine as if it were entered on a regular keyboard. Positional input by means of cell selections keys is translated into its equivalent semantic PUIR event, and presented to the AUI engine.

Regardless of the delivery path, input is processed by the AUI engine, and any results are broadcast to rendering agents (and often the application) by means of events.

5 Testing

The design and implementation of the PUIR framework has been tested throughout its entire development cycle, both in terms of functionality and correctness.

Basic unit testing was conducted by comparing the functionality of UI test cases implemented both using the PUIR framework and as Java Swing applications. This testing has proven to be extremely important in view of the complexity of interfacing with existing user interface toolkits (such as Java Swing), and in terms of verifying the accuracy of the chosen abstractions.

The project is currently entering the stage of real user testing, to assess the effectiveness of the PUIR framework in providing blind users access to applications with graphical user interfaces, and to assess whether it successfully promotes effective collaboration between users.

6 Conclusion

Parallel user interface rendering is proving to be a very powerful technique in support of the Design-for-All principle. It builds on a solid base of research and development of abstract user interfaces, and provides a framework where alternative renderings of the UI operate at the same level as the native rendering rather than being a derivate. The coherence between the renderings supports close collaboration between users because they can communicate about interacting with an application based on a substantially similar mental model of the user interface.

The PUIR framework can contribute to the field of accessibility well beyond the immediate goal of providing non-visual 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. functional access to GUI applications.

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. Weber, G., Mager, R.: Non-visual user interfaces for X Windows. In: ICCHP 1996: Interdisciplinary aspects on computers helping people with special needs, pp. 459–468 (1996)
2. Haneman, B., Mulcahy, M.: The GNOME accessibility architecture in detail. Presented at the CSUN Conference on Technology and Disabilities (2002)
3. Gajos, K., Weld, D.S.: SUPPLE: automatically generating user interfaces. In: IUI 2004: Proceedings of the 9th international conference on Intelligent user interfaces, pp. 93–100. ACM Press, New York (2004)

4. 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)
5. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M.: UsiXML: A user interface description language supporting multiple levels of independence. In: Lauff, M. (ed.) *Proceedings of Workshop on Device Independent Web Engineering, DIWE 2004* (2004)
6. Kost, S.: Dynamically generated multi-modal application interfaces. PhD thesis, Technische Universität Dresden, Dresden, Germany (2006)
7. Van Hees, K., Engelen, J.: Abstract UIs as a long-term solution for non-visual access to GUIs. In: *Proceedings of the 3rd International Conference on Universal Access in Human-Computer Interaction* (2005)
8. Van Hees, K., Engelen, J.: Abstracting the graphical user interface for non-visual access. In: Pruski, A., Knops, H. (eds.) *Assistive technology from virtually to reality, 8th European conference for the Advancement of Assistive Technology in Europe*, pp. 239–245. IOS Press, Amsterdam (2005)
9. Van Hees, K., Engelen, J.: Non-visual access to GUIs: Leveraging abstract user interfaces. In: Miesenberger, K., Klaus, J., Zagler, W.L., Karshmer, A.I. (eds.) *ICCHP 2006. LNCS, vol. 4061*, pp. 1063–1070. Springer, Heidelberg (2006)
10. Mynatt, E.D., Weber, G.: Nonvisual presentation of graphical user interfaces: contrasting two approaches. In: *CHI 1994: Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 166–172. ACM Press, New York (1994)
11. Gunzenhäuser, R., Weber, G.: Graphical user interfaces for blind people. In: *13th World Computer Congress*, pp. 450–457 (1994)
12. Savidis, A., Stephanidis, C.: Building non-visual interaction through the development of the rooms metaphor. In: *CHI 1995: Conference companion on Human factors in computing systems*, pp. 244–245. ACM Press, New York (1995)
13. Savidis, A., Stergiou, A., Stephanidis, C.: Generic containers for metaphor fusion in non-visual interaction: the HAWK interface toolkit. In: *Proceedings of the Interfaces 1997 Conference*, pp. 194–196 (1997)
14. Rose, D., Stegmaier, S., Reina, G., Weiskopf, D., Ertl, T.: Non-invasive adaptation of black-box user interfaces. In: *AUIC 2003: Proceedings of the Fourth Australasian user interface conference on User interfaces 2003*, pp. 19–24. Australian Computer Society, Inc. (2003)
15. Barnicle, K.: Usability testing with screen reading technology in a Windows environment. In: *CUU 2000: Proceedings on the 2000 conference on Universal Usability*, pp. 102–109. ACM Press, New York (2000)
16. Pontelli, E., Gillan, D., Xiong, W., Saad, E., Gupta, G., Karshmer, A.I.: Navigation of HTML tables, frames, and XML fragments. In: *Assets 2002: Proceedings of the fifth international ACM conference on Assistive technologies*, pp. 25–32. ACM Press, New York (2002)
17. Theofanos, M.F., Redish, J.G.: Bridging the gap: between accessibility and usability. *Interactions* 10(6), 36–51 (2003)
18. Trewin, S., Zimmermann, G., Vanderheiden, G.: Abstract user interface representations: how well do they support universal access? In: *CUU 2003: Proceedings of the 2003 conference on Universal usability*, pp. 77–84. ACM Press, New York (2003)
19. Bass, L., Faneuf, R., Little, R., Mayer, N., Pellegrino, B., Reed, S., Seacord, R., Sheppard, S., Szczur, M.R.: A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop. *SIGCHI Bull.* 24(1), 32–37 (1992)