Book Title
Book Editors
IOS Press, 2003

Abstracting the Graphical User Interface for Non-Visual Access

Kris Van Hees, Jan Engelen
Katholieke Universiteit Leuven
kris@alchar.org, jan@docarch.be

Abstract. Providing blind users with access to graphical user interfaces is still a very complex problem. Existing screen readers face many obstacles that stem from the fact that environments are usually not designed with accessibility in mind. Expanding upon many advances that have emerged in recent years in support of the Design For All principle, an alternative approach to providing non-visual access to graphical user interfaces is presented.

Keywords. accessibility, abstract user interface, screen reader, HCI

1. Introduction

The emergence of graphical user interfaces (GUIs) caused quite a concern within the community of blind users [1]. While text-based screens were mostly accessible to screen readers, the new technology presented a whole new range of challenges. Sighted users usually consider a GUI to be more intuitive, whereas blind users are more commonly hindered by the use of primarily visual concepts.

The popularity of MS Windows-based systems in the workplace and at home, and the existence of commercial screen readers on this platform caused somewhat of an artificial comfort zone. Some felt that the crisis had been averted. Other systems such as Unix remained a concern, but these were often considered to be very specialised environments. Remote access to Unix systems from MS Windows workstations was considered sufficient.

The increase in popularity of Unix systems, both in work and home environments, has shifted the balance in recent years, raising the priority on providing non-visual access to GUIs on Unix. Mynatt and Weber describe two early approaches: Mercator and GUIB [2].

Edwards, Mynatt and Stockton published a paper as early as 1994 [3], stressing the importance of providing access to GUIs rather rather than graphical screens, thereby setting the stage for using non-visual representations rather than trying to interpret the visual image of windows. More recent research into this area focuses on the specific obstacles that blind users encounter with GUIs [4], spatial auditory interfaces [5], and multi-modal interfaces[6].

The similarity between web forms and user interfaces (UIs) drives the research presented in this paper. Research into web accessibility [7,8,9] and abstract user interfaces

(AUIs) [10,11] is combined to provide an alternative approach to non-visual access to GUIs.

The remainder of this paper first presents related work on GUI accessibility. The third section describes the use of an AUI at the core of an accessibility framework, while the fourth section focuses on the integration with existing AUI applications. The fifth section concludes this paper with a description of future work.

2. Related Work

User interfaces are a very important topic within the realm of Human-Computer Interactions. For the purposes of this paper, the work done at the Belgian Laboratory of Computer-Human Interaction (BCHI) at the Université Catholique de Louvain is of great importance [12]. The ability to abstract the user interface of applications lies at the core of the methods proposed in this paper.

The “Fruit” system [10] described by Kawai, Aida, and Saito addresses the issue of UI accessibility as well. Rather than using an AUI, an abstract widget toolkit is used. The presentation of the UI is handled by a device-specific rendering component. The “Fruit” system primarily deals with representing the UI on any one arbitrary device that is supported by the system. Two aspects of accessibility that are not present in the system are synchronised presentation in multiple modalities (e.g. presenting the UI both visually and auditorily to facilitate cooperation between sighted and blind users) and accessibility at the windowing system level to handle window management functionality.

The Visualisation and Interactive Systems Group of the University of Stuttgart described a system for black-box UIs [13]. Their work was aimed at replacing a user interface by means of interposing libraries. The presented solution for non-invasive adaptation of UIs is useful for capturing widget toolkit function calls for testing purposes. It could also be used to enable legacy applications to use improved or adapted versions of a given UI toolkit.

3. Accessibility and AUIs

One of the more complex aspects of GUIs on Unix systems is the quite common practice of simultaneously using applications built upon different widget toolkits. Screen readers on Unix therefore must handle the different widget toolkits. Figure 1 shows parts of three different application UIs: Firefox, XFig, and GAIM. The top snapshots both contain a menu bar, while all three contain buttons. While the look of the menu bars and the buttons is quite distinctly different between the snapshots, users intuitively know that all menu bars essentially work the same way. The same applies to the buttons. Functionality is the common factor.

The Gnome desktop environment [14] provides accessibility for various widget toolkits provided that the toolkits implement an “accessibility bridge” that can interface with the Assistive Technology Service Provider Interface (AT-SPI). Haneman and Mulcahy provide a detailed architecture [15] diagram for the Gnome Accessibility Architecture, describing this dependency (see Figure 2). The application layer in the diagram covers both the actual application code and the widget toolkit implementation. Applica-

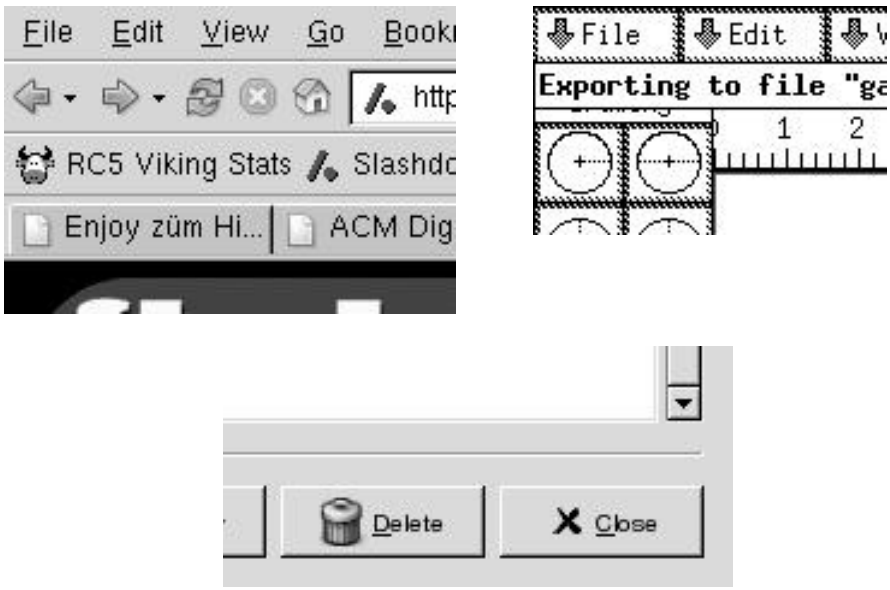

Figure 1. Parts of UIs using various widget toolkits: Firefox (upper left), XFig (upper right), GAIM (bottom)

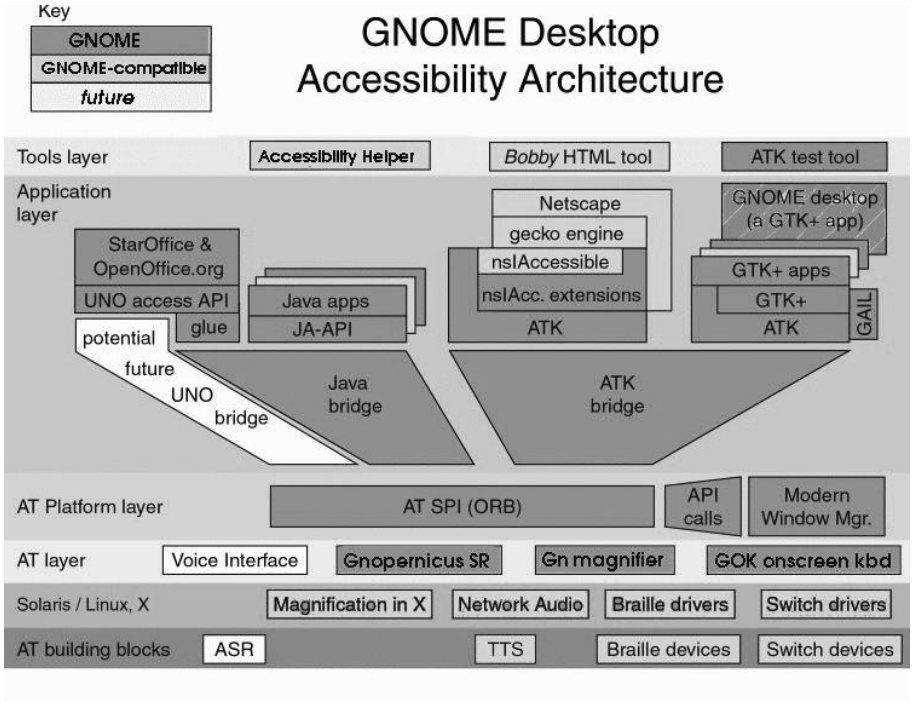

Figure 2. The GNOME Desktop Accessibility Architecture

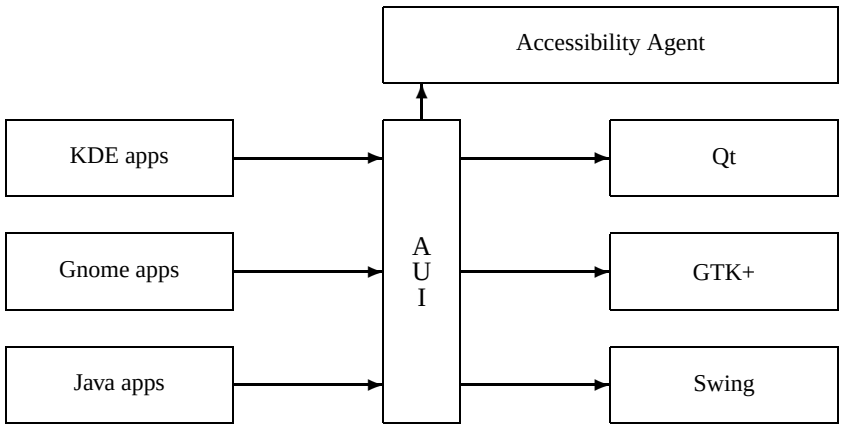

Figure 3. Abstracting the UI: Schematic overview

tions must be implemented with accessibility in mind, by calling specific functions in the widget toolkit.

The well known development paradigm to separate presentation and application logic makes it possible to avoid the widget toolkit specific accessibility bridge. Rather than constructing the UI by means of function calls into a specific widget toolkit, applications will implement their UI by means of an AUI definition. This definition can be constructed programmatically using specific UI creation tools that generate a description of the UI in a standard format (e.g. UsiXML [12]). The visualisation of the UI is then delegated to widget toolkit specific AUI interpreters, while non-visual presentation of the UI is be handled by alternative AUI interpreters that can serve screen readers. Figure 3 provides a schematic description of this approach. For this technique to be successful, the AUI must be able to represent both data and a description on how to present that data, as suggested by Trewin, Zimmermann, and Vanderheiden [11].

Mynatt and Weber discuss four important HCI design issues [2] that need to be addressed as part of any non-visual presentation of GUIs:

- Coherence between visual and non-visual interfaces
  Collaboration between sighted and blind users requires coherence between the visual and non-visual interfaces. The mental model of the UI must be substantially similar to both user groups to allow clear communication about how to use an application to accomplish a specific goal, and one user should be able to observe the actions of another.
  By abstracting the UI and visualising it by means of specific AUI interpreters coherence is guaranteed. Both the visual and the non-visual presentations are generated from a single source that provides both a description of the UI and the data presented in it.
- Exploration in a non-visual interface
  Non-visual modalities (auditory and tactile) are limited in their ability to provide information to the user in part due to their largely serial nature, whereas a visual UI can provide information in parallel in a very efficient way.
  The screen reader implementation must provide specific non-visual mechanisms to explore the non-visual interface. Given that the GUI is capable of providing

information by means of spatial properties of UI elements, often beyond the scope of a single application, non-visual alternatives must also be provided.

- Conveying graphical information in a non-visual interface
  GUIs commonly present information in a strict visual way: icons, attributes on UI elements, appearance of UI elements, ...
  Due to the nature of the AUI mechanism, visual aspects of UI elements are handled by the AUI interpreters. The non-visual interpreter can provide information to the screen reader concerning the properties of UI elements. It is the responsibility of the screen reader to present this information in a usable way.
- Interaction in a non-visual interface
  Interaction in a GUI is often based on visual idioms (clicking a button, moving a slider, ...) whereas a blind user requires specific non-visual forms of interaction. The screen reader is responsible for providing modes of interaction that can be translated into their equivalent visual modes. The AUI component is therefore not only responsible for the presentation of the UI, but also for abstracting user input (where needed). An example of this functionality would be translating specific key combinations into mouse operations.

An additional advantage of being able to interpret an AUI in a non-visual way can be found in centralised computing environments. Executing an application on a remote system and displaying the UI on the local system is a very common practise in the Unix-world. Programmatically defined GUIs typically require the screen reader to issue function calls across the network in order to query attributes of specific UI objects. A screen reader based on an AUI could request the current state of (part of) the UI in a single remote function call and interpret it locally. Experiments will show whether this is a significant advantage.

4. Integration with existing AUIs

While this paper proposes abstracting the UI of applications as a solution to providing non-visual access to application, it is unlikely that developers will be interested in adopting such a technique purely for the purpose of supporting accessibility. On the other hand, the potential for user-selectable “look and feel” by specifying which visualisation should be used may be an attractive feature of this technique.

Past [10] and current [11,16] research on abstract user interfaces expands upon the separation of presentation and application logic, and the observation that the talent required to develop UIs is quite different from the talent required to write application logic. An area that receives a lot of attention is the development of UI creation tools, generating an abstract description of the UI. Rather than trying to provide multiple layers of abstraction, it is proposed that existing UI description languages (such as UsiXML) are extended to provide the information needed for non-visual presentation.

5. Conclusion

This paper presents an alternative technique for providing non-visual access to graphical user interfaces, by means of an abstraction of the user interface. While the technique

is theoretically feasible, only an actual experimental implementation can truly put it to the test. In the coming months, an initial AUI description will be defined along with the implementation of a visual rendering agent based on existing widget toolkits. Building upon that work, a non-visual rendering agent will be implemented together with a basic screen reader. Throughout all research and development, experimental implementations will be presented to blind users for feedback on the techniques used and its effectiveness.

Various questions remain unanswered, and will require additional research in coming months. The interpretation of the AUI for non-visual presentation requires mappings from mostly visual metaphors to non-visual one. Proper integration with existing AUI frameworks is required, because this technique heavily depends on the adoption of the AUI application development paradigm.

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] L. H. Boyd, W. L. Boyd, and G. C. Vanderheiden. The graphical user interface: Crisis, danger and opportunity. In *Journal of Visual Impairment and Blindness*, pages 496–502, 1990.

[2] Elizabeth D. Mynatt and Gerhard Weber. Nonvisual presentation of graphical user interfaces: Contrasting two approaches. In *Human Factors in Computing Systems*, pages 166–172. CHI 94 – Celebrating Interdependence, 1994.

[3] W. Keith Edwards, Elizabeth D. Mynatt, and Kathryn Stockton. Providing access to graphical user interfaces – not graphical screens. In *Assets '94: Proceedings of the first annual ACM conference on Assistive technologies*, pages 47–54. ACM Press, 1994.

[4] Kitch Barnicle. Usability testing with screen reading technology in a windows environment. In *CUU '00: Proceedings on the 2000 conference on Universal Usability*, pages 102–109. ACM Press, 2000.

[5] Hilko Donker, Palle Klante, and Peter Gorny. The design of auditory user interfaces for blind users. In *NordiCHI '02: Proceedings of the second Nordic conference on Human-computer interaction*, pages 149–156. ACM Press, 2002.

[6] Holly S. Vitense, Julie A. Jacko, and V. Kathlene Emery. Multimodal feedback: establishing a performance baseline for improved access by individuals with visual impairments. In *Assets '02: Proceedings of the fifth international ACM conference on Assistive technologies*, pages 49–56. ACM press, 2002.

[7] Thomas Kieninger. The "growing up" of hyperbraille – an office workspace for blind people. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 67–73. ACM press, 1996.

[8] E. Pontelli, D. Gillan, W. Xiong, E. Saad, G. Gupta, and A. I. Karshmer. Navigation of html tables, frames, and xml fragments. In *Assets '02: Proceedings of the fifth international ACM conference on Assistive technologies*, pages 25–32. ACM Press, 2002.

[9] Mary Frances Theofanos and Janice Redish. Bridging the gap: between accessibility and usability. *Interactions*, 10(6):36–51, 2003.

[10] Shiro Kawai, Hitoshi Aida, and Tadao Saito. Designing interface toolkit with dynamic selectable modality. In *Assets '96: Proceedings of the second annual ACM conference on Assistive technologies*, pages 72–79. ACM Press, 1996.

[11] Shari Trewin, Gottfried Zimmermann, and Gregg Vanderheiden. Abstract user interface representations: How well do they support universal access? In *CUU '03: Proceedings of the 2003 conference on Universal usability*, pages 77–84. ACM Press, 2003.

[12] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, and M. Florins. Usixml: A user interface description language supporting multiple levels of independence. In M. Lauff, editor, *Proceedings of Workshop on Device Independent Web Engineering DIWE'04 (Munich, 26-27 July 2004)*, 2004.

[13] D. Rose, S. Stegmaier, G. Reina, D. Weiskopf, and T. Ertl. Non-invasive adaptation of black-box user interfaces. In *CRPITS '18: Proceedings of the Fourth Australian user interface conference on User interfaces 2003*, pages 19–24. Australian Computer Society, Inc., 2002.

[14] Russell Dyer. The gnome 2 desktop environment. *Linux Journal*, 2003(108):7, 2003.

[15] Bill Haneman and Marc Mulcahy. The gnome accessibility architecture in detail. Available at http://developer.gnome.org/projects/gap/presentations/, 2002. Presented at the CSUN Conference on Technology and Disabilities; GNOME Architecture diagram used with author's permission.

[16] Julien Stocq and Jean Vanderdonckt. A domain model-driven approach for producing user interfaces to multi-platform information systems. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 395–398. ACM Press, 2004.