Non-visual access to GUIs: Leveraging abstract user interfaces

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. Various approaches to providing blind users with access to graphical user interfaces have been researched extensively in the past 15 years, and yet accessibility is still facing many obstacles. Graphical environments such as X Windows offer a high degree of freedom to both the developer and the user, complicating the accessibility problem even more. Existing technology is largely based on either a combination of graphical toolkit hooks, queries to the application and scripting, or model-driven user interface development. Both approaches have limitations that the proposed research addresses. This paper builds upon past and current research into accessibility, and promotes the use of abstract user interfaces to providing non-visual access to GUIs.

1 Introduction

Ever since graphical user interfaces (GUIs) emerged, the community of blind users has been concerned about its effects on computer accessibility [1]. Until then, screen readers were able to truly read the contents of the screen and render it in tactile and/or audio format. GUIs introduced an inherently visual interaction model with a wider variety of interface objects. MS Windows rapidly became the *de facto* graphical environment of choice at home and in the workplace because of its largely consistent presentation, and the availability of commercial screen readers created somewhat of a comfort zone for blind users who were otherwise excluded from accessing GUIs.

As X Windows-based systems grow in popularity for both home and work environments, an additional level of complexity has emerged. Not only pose GUIs an obstacle by being inherently visual, but it is also possible to combine elements from a variety of graphical toolkits such as Athena, GTK, Qt, . . . into a single graphical environment. This important feature promotes flexibility and interoperability, but it largely complicates the work needed to provide accessibility. Screen readers either must support all commonly used toolkits, or they must be designed to not depend on any implementation specific details.

Past and current research indicates that abstracting the user interface offers a high degree of flexibility in rendering for a multitude of output modalities. Blind users generally prefer auditory and/or tactile representations of the GUI, as an alternative to the visual rendering provided by GUIs. Current approaches use a combination of toolkit extensions, scripting, and complex heuristics to obtain sufficient information to make alternative renderings possible[2]. Alternative approaches aim to solve the accessibility problem by addressing the different output modalities in the design and development of applications, composing model-driven user interface implementations at development time.

The need for alternative user interface representations across different output modalities makes the accessibility problem a prime candidate for using abstract user interface (AUI) descriptions. The remainder of this paper first presents related work on GUI accessibility. The third section describes the use of AUIs at the core of an accessibility framework, providing non-visual rendering in parallel with visual representations, followed by a comparison of this approach against AUI-based model-driven user interface construction. Section five concludes this paper with a description of future work.

2 Related Work

In the context of accessibility of GUIs for blind users, Mynatt and Weber provided two early approaches[3]. The Mercator project at the Georgia institute of Technology replaced the GUI with a hierarchical auditory interface, whereas GUIB provided a tactile representation of the screen contents. Contrasting both projects also established four core design issues that are common to non-visual access to GUIs, further refined in [4]. This early paper also provides a more general description of common approaches towards GUI accessibility.

The ”Fruit” system described by Kawai, Aida, and Saito[5] addresses the issue of user interface accessibility by means of an abstract widget toolkit. Application software is still written as if a graphical widget toolkit is being used, while the actual presentation of the user interface is handled by device-specific components. The ”Fruit” system does not support synchronised presentation in multiple modalities, nor does it provide any accessibility at the level of the windowing environment.

Savidis and Stephanidis explored alternative interaction metaphors for non-visual user interfaces[6]. This work was expanded upon in the development of a user interface development toolkit[7]. The HAWK toolkit provides interaction objects and techniques that have been designed specifically for non-visual access. The toolkit is used in the AVANTI project[8], introducing a Unified User Interface concept using runtime user interface adaptation based on user and usage context.

The similarities between application user interfaces and World Wide Web forms provide an important foundation for using AUIs. Barnicle researched specific obstacles that blind users encounter when using GUIs[12]. His results were confirmed in later research[13, 14]. The UsiXML[9] project at the Belgian Laboratory of Computer-Human Interaction (BCHI) at the Université Catholique de Louvain builds upon these concepts. The ability to abstract the user interface lies at the core of the research presented in this paper.

Research into alternative user interfaces has been extensive. Two notable projects are the ”virtual sound wall” at the Oldenburg R&D-Institute for Informatics Tools and Systems[10], and the performance analysis of multi-modal interfaces by Vitense, Jacko, and Emery[11]. Both solutions require expensive devices that are well outside the budget of an average user. The research presented in this paper therefore limits the context of output modalities to refreshable Braille displays, speech synthesisers, and/or non-spatial sound.

3 Leveraging AUIs towards accessibility

The non-visual presentation of GUIs poses several HCI design issues that need to be addressed as part of any acceptable accessibility solution. Five fundamental problems were identified by Mynatt and Weber[3], and Gunzenhäuser and Weber[4]:

- 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

Abstract user interface rendering at runtime ensures coherence, and allows both sighted and blind users to operate using the same mental model of the interaction semantics[15, 16]. In addition, AUIs can address the three remaining fundamental HCI problems by translating what is perceived as visual metaphors in a representation independent manner. Some of this task is to be delegated to the screen reader implementation. E.g. while the AUI rendering is handled per application by shared components, the screen reader is typically implemented as an application that serves the entire windowing environment. Figure 1 explains how non-visual access to GUIs can be implemented by leveraging AUIs.

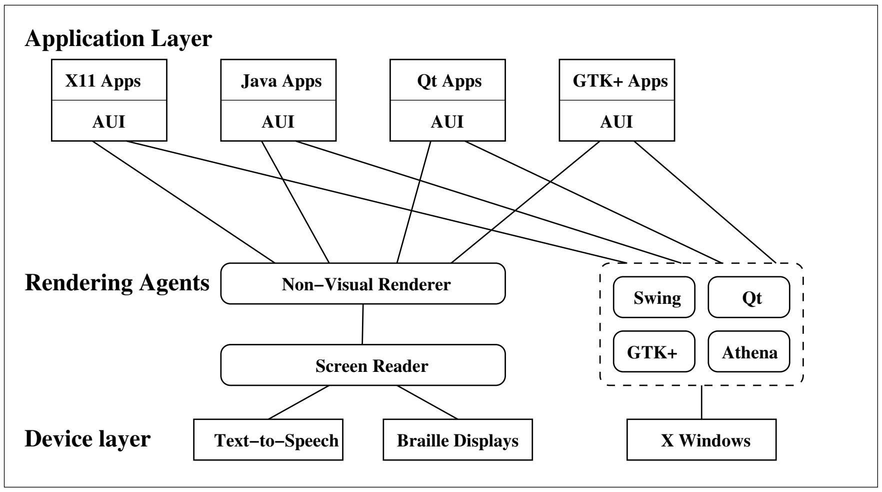

Fig. 1. Leveraging abstract user interfaces for accessibility

Whereas current approaches work with applications that are implemented against a specific graphical toolkit, depending on explicit support for accessibility in the toolkit, AUI-based non-visual access builds on a paradigm where the UI of the application is described in an abstract form. The presentation of the UI is delegated to specific rendering agents, both visual and non-visual[15].

The advantages of this approach are not only within the context of providing non-visual access to GUIs but also within the context of user-controlled "look & feel". Because applications are no longer specifically built for a given graphical toolkit and rendering is delegated to specific agents, it is possible to render applications against any supported toolkit. This flexibility can be a powerful feature for many users.

Leveraging AUIs allows accessible user interfaces to be implemented side-to-side with their graphical counterparts. This resolves a long standing problem with screen readers needing to tap into the application flow in order to retrieve the information needed to drive alternative presentations. This puts current screen reading technologies at a definite disadvantage.

The XML document shown in figure 2 provides an example of a fairly simple user interface. The graphical rendering in Swing is shown in figure 3. The example application is a card game, appealing to both a sighted and blind audience.

```
<?xml version="1.0"?>
<gui>
  <window id="GameSys">
    <menuBar>
      <menu id="Game">
        <menuItem id="New" label="New game"/>
        <menuSeparator/>
        <menuItem id="Quit"/>
      </menu>
      ...
    </menuBar>
    <form id="Table" width="500" height="300"
          bgImage="felt.jpg">
      <form id="Card-1" width="71" height="96"
            bgImage="card-1.gif"/>
      ...
    </form>
    <statusBar label="GameSys v1.0.0"/>
  </window>
</gui>
```

Fig. 2. XML document providing an abstract user interface description

While the example presented here is very basic, it does show important features of the presented approach. The menu bar with all its components is defined in a truly

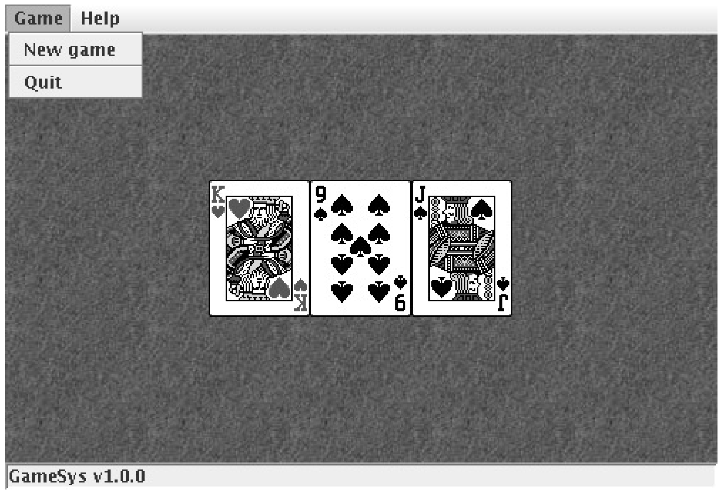

Fig. 3. Rendering an AUI description using Swing

abstract way, whereas the main user interface uses an abstract form augmented with additional (graphical) information, such as image sizes and background image specifications. It carries all information to facilitate both visual and non-visual presentations. While the example is a hand-crafted AUI description, more complicated applications would benefit from generated user interface descriptions (e.g. using UsiXML[9]).

The reference implementation currently being developed parses the XML document describing the user interface into an abstract object model. This model drives all rendering engines, providing them with information about what needs to be communicated to the user. The rendering engines decide how the information is presented. Input from the user is channelled from the input device to the focus manager in the abstract object model, interacting both with the application and with the rendering engines. The current architecture for the prototype is shown in figure 4. Note that whereas the XML document describes only how the user interface is presented to the user, the abstract object model also carries information on what data is to be presented as is suggested by earlier research[17].

Although Swing is used as the underlying technology for this prototype, all of the UI logic is handled by the AUI focus manager. This ensures that both the visual and the non-visual presentations provide users with identical interaction models. In addition, the focus manager can make decisions based on graphical information (element sizes, etc…) without exposing the actual decision logic to the rendering agents. This is required in order to resolve the problem of conveying graphical information in a non-visual interface.

4 Comparison with model-based UI construction

Model-based UI construction generally covers two approaches:

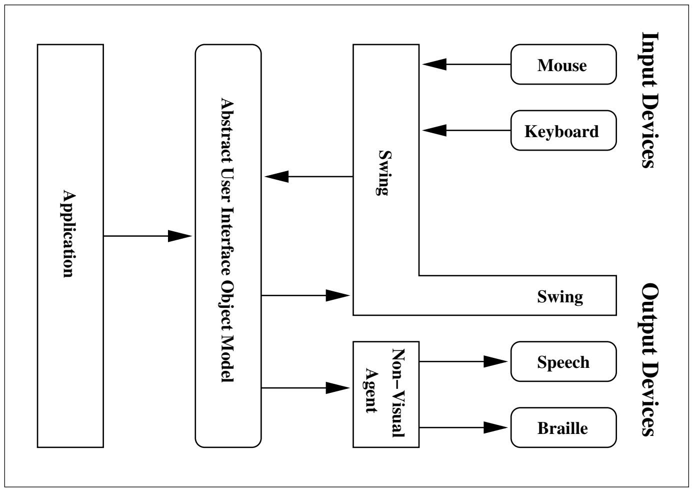

Fig. 4. Reference implementation architecture

– Development-time UI construction
  This includes all ways of generating multiple final user interfaces (generally for a variety of modalities). Often, rather than simply creating multiple UI front-ends, multiple versions of the application are generated due to using a more invasive interface between the application and the user interface. A prime example is the UsiXML project (and its derived projects)[9].
– Runtime UI selection
  This includes all mechanisms that allow some form of model-driven UI customisation. Where development-time UI construction generates a set of UIs, runtime UI selection supports alternatives for UI components. A model-driven UI engine determines which alternative (if any) for a given component is to be used at any given time. A good example of this approach is the AVANTI project[8].

The research presented in this and earlier papers[15, 16] can and will build on development-time UI construction techniques, but it present a very different approach in terms of accessibility. Providing non-visual access to UIs by means of modality specific implementations constructed during the development of the application makes it impossible to share a single instance between a sighted and a blind user. As such, collaboration by observing the same runtime information is not possible. Runtime rendering of the AUI allows for this by supporting multiple simultaneous renderings.

Runtime UI selection is somewhat similar to the approach presented here, in the sense that the user interface presentation is decided upon at runtime rather than at development time. This powerful technique supports current concepts of universal access, and is a major advancement towards accessibility. It does not generally provide for simultaneous rendering in different modalities. It does involve a tight coupling between the application and the user interface presentation, making future extension of supported modalities more complicated. Runtime rendering of AUIs does not have that limitation.

5 Conclusion

This paper describes a novel approach to providing a long-term solution for non-visual access to GUIs by leveraging abstract user interfaces. The theory behind the presented work is built upon extensive research into HCI accessibility issues, AUIs, past and current approaches to solving this complex problem, and user feedback on existing implementations.

A prototype is being developed for extensive field testing as part of this research. Only through user feedback can success be measured appropriately. The prototype will be enhanced to provide better support for a representative subset of commonly used widgets so that multiple example applications can be built.

There is still a long way ahead, and ultimately, integration with existing (and in-development) AUI frameworks will be important. Not only because modifications may be required in order to support runtime rendering and non-visual access, but also because the proposed approach builds upon the adoption of the AUI application development paradigm. because

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. Gerhard Weber and Rolf Mager. Non-visual user interfaces for X Windows. In *Interdisciplinary aspects on computers helping people with special needs*, pages 459–468. 5th International Conference, ICCHP 96, 1996.
3. 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.
4. Rul Gunzenhäuser and Gerhard Weber. Graphical user interfaces for blind people. In Brunnstein K. and E. Raubold, editors, *13th World Computer Congress 94, Volume 2*, pages 450–457. Elsevier Science B.V., 1994.
5. Shiro Kawai, Hitoshi Aida, and Tadao Saito. Designing interface toolkit with dynamic selectable modality. In *Proceedings of the second annual ACM conference on Assistive Technologies*, pages 72–79. ACM Press, 1996.
6. Anthony Savidis and Constantine Stephanidis. Building non-visual interaction through the development of the Rooms metaphor. In companion of the *CHI'95 conference in Human Factors in Computing Systems*, pages 244–245, 1995.
7. Anthony Savidis, Athena Stergiou, and Constantine Stephanidis. Generic Containers for Metaphor Fusion in Non-Visual Interaction: the HAWK Interface Toolkit. In *Proceedings of the Interfaces '97 Conference*, pages 194–196, 1997.

8. Constantine Stephanidis and Anthony Savidis. Universal Access in the Information Society: Methods, Tools, and Interaction Technologies. In Reinhard Oppermann, editor, *Universal Access in the Information Society*, 1(1):40-55, 2001.
9. 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.
10. Hilko Donker, Palle Klante, and Peter Gorny. The design of auditory user interfaces for blind users. In *Proceedings of the second Nordic conference on Human- Computer Interaction*, pages 149–156. ACM Press, 2002.
11. 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 *Proceedings of the fifth international ACM conference on Assistive Technologies*, pages 49–56. ACM Press, 2002.
12. Kitch Barnicle. Usability testing with screen reading technology in a windows environment. In *Proceedings on the 200 conference on Universal Usability*, pages 102–109. ACM Press, 2000.
13. E. Pontelli, D. Gillan, W. Xiong, E. Saad, G. Gupta, and A. I. Karshmer. Navigation of html tables, frames, and xml fragments. In *Proceedings of the fifth international ACM conference on Assistive Technologies*, pages 25–32. ACM Press, 2002.
14. Mary Frances Theofanos and Janice Redish. Bridging the gap: between accessibility and usability. *Interactions*, 10(6):36–51, 2003.
15. Kris Van Hees and Jan Engelen. Abstract UIs as a long-term solution for non-visual access to GUIs. In *Proceedings of the 11th International Conference on Human-Computer Interaction* (CD-ROM), 2005.
16. Kris Van Hees and Jan Engelen. Abstracting the Graphical User Interface for Non-Visual Access. In A. Pruski and H. Knops, editors, *Assistive Technology: From Virtuality to Reality (AAATE 2005)*. IOS Press, 2005.
17. 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.