Designing an OffScreen Model for a GUI

Dirk Kochanek

Universität Stuttgart, Institut für Informatik
Breitwiesenstraße 20-22, D-70565 Stuttgart
F. R. Germany

Abstract. Blind or visually impaired people get access to computers by using a screen reader. A screen reader is a special access program that operates on a database modelling the user interface and presents it in an appropriate form to the user. While in textbased user interfaces this database is fairly simple for graphical user interfaces a more sophisticated design is necessary. This article describes the design issues of such a database as well as an overview of a complete architecture of a screen reader system.

Introduction

For a long time access to GUIs was neither considered necessary nor possible [1]. This was on one hand caused by the high price of machines able to run GUIs. On the other hand the obstacle built up by presenting text as graphics (pixels) seemed to be insuperable.

With the advancement of technology and the accompanying breakdown in prices the use of GUIs became more and more popular. This trend is also reflected by today's applications. A survey among leading software producers conducted in 1992 has shown that 30% are only producing GUI-based software. 93% are producing some GUI-based software while only 27% plan to have text-based alternatives to the GUI version [2]. These figures show the strong need for suitable adaptations of graphical user interfaces for persons who are blind.

The complexity of a GUI due to its graphical nature is putting an additional mental load on blind users. Using a GUI brings also some advantages. All applications running on a GUI will mainly use interaction objects provided by the interface. For consistency interaction objects share the same lock and feel. This uniformity can not be achieved easily in textbased applications.

Screen Readers

A screen reader is a special program giving access to computer systems that are otherwise inaccessible to blind or partially sighted users. It will present its output in a form appropriate for the users.

A screen reader operates in two modes: tracking mode and review mode. In tracking mode the screen reader follows every change occurring in an application. These changes can be caused by an external event (i.e. arriving mail) of by actions taken by the user (i.e. keyboard). In review mode the user can explore the screen independent

from applications currently running. The user has to therefore either explicitly enter a special review mode using the standard keyboard or implicitly by using dedicated keys or a separate keyboard.

Different media can be used to convey the information to the user. Synthetic speech is a quick and cost effective solution while refreshable braille is a more expensive medium but offers a unsurpassed notational precision. 2D or even 3D nonspeech sounds can convey information about status of positions that are otherwise difficult to present. Although most screen readers are designed to use only one output medium users will benefit from combining two or more.

The Pixel Barrier

Access programs for textbased applications are usually driven by events such as keyboard actions and changes of the screen content. Keyboard actions can be detected by hooking into the keyboard service of the operating system. Changes of the screen content are either detectable by hooking the video service of the operating system or by constantly monitoring the video (refresh) memory. Video memory is used as a form of database for retrieving text from the screen.

Access programs operating on a GUI rely on events that have to be filtered within the GUI. Monitoring changes of the video memory is, compared to a textbased UI, a time consuming task due to the vast amount of data that has to be compared (> 256K for a GUI vs. 4K in a textbased UI). On the other hand the video memory is no longer an easy source for text retrieval because text is no longer stored as character/attribute pairs. Instead every character is broken into several pixels. These pixels are points of different colour and intensity.

This „pixel barrier" can be overcome by introducing a virtual screen copy (VISC). The VISC is a database for information about every single pixel to what character it belongs to. This information can be gained in two ways. First by intercepting the process when a character is broken into pixels. This can for example be done by modifying the video driver. Second in reversing the pixel process by applying OCR techniques to either the video memory or directly to the video signal. While the last method has the advantage to be completely independent of the operating system and GUI it is difficult to achieve high accuracy and/or good performance [3] [4] [5].

The second problem that comes up when adapting a GUI is its use of high level interaction objects such as windows, menus, buttons and listboxes. Since the VISC will contain only information on a lexical level such objects are difficult to detect or discriminate. Several applications running concurrently are causing another problem. Windows of such applications can overlap making text hard to read. One solution would be to restrict the scope of presentation to the active application. Since the VISC is lacking such syntactic information as windows dimensions overlapping windows can not be separated.

Instead a database that modeling this structure is required. Such a database is then called offscreen model or OSM. It is also advisable to incorporate information kept in the VISC into this database.

Interaction objects

The elements of a GUI are called interaction objects. Basic objects are windows, a rectangular region with a frame around it, and icons, a symbol for a minimised application. Other interaction objects allow the selection of items from a list. In case the list has a fixed number of items the object is called menu, for an undefined number listbox. Actions or settings can be triggered by various forms of buttons. We can distinguish between pushbuttons, checkboxes, radiobuttons etc. Objects containing descriptive text are called labels or statics while those with changeable text are called text- or editfields. Settings within a certain range can be represented by sliders or scrollbars.

While the appearance and mode of operation of a class of interaction objects vary between GUIs the standard functionality of a class as such remains the same.

Design issues

When designed carefully an OSM should be suitable for different GUIs. This means it should be independent of the underlying operating system and/or GUI. It should be furthermore invariant from the source of information. So it should not matter whether text is retrieved by means of OCR or a modified video driver.

Taking care of these design issues a screen reader operating on an OSM will to a high degree be independent from the GUI. To simplify development and maintenance of a screen reader another level of abstraction can be introduced.

Sources of Information

Since the OSM is a specialised database the question rises how to fill its entries. Two methods of filtering text were already mentioned above (OCR and modified video driver). The source for syntactic information is the GUI itself. By using so-called hooks it is possible to get access to internal information within the GUI. This is a „legal" method since hooks are provided by the manufacturer of the GUI. On the other side access is somehow restricted since the hooks are often not designed to filter information for an OSM.

Another possible way is by directly modifying the GUIs' executable by means of patching. Patching can be prohibited due to security reasons since it changes pointers to functions which is in most operating systems an illegal since impossible operation because GUI and filter run in different address spaces. In systems with lower security like MS-Windows 3.1 where patching is possible however it allows access to nearly every piece of information that is considered necessary or has otherwise to be deducted from other sources in a more or less complicated way.

To simplify development of applications more and more user interface toolkits are used. By modifying or extending such a toolkit events for the screen reader as well as all necessary information to keep the OSM up-to-date can be deducted. Although this filter approach is only valid for new applications build with such a modified toolkit it is a very promising way since the information available is of first hand.

OSM

Nearly all GUIs are internally using a tree-like structure to represent interaction objects and their dependencies. Therefore it is only logical to choose a n^{th} tree as basic data structure for the OSM since a tree best maps the internal structure. It also implicitly incorporates parent-child relations between interaction objects and is also independent of the type of information kept in one node.

The screen reader needs to perform certain operations on the OSM. These operations can be identified by the following methods:

instanciate: create and destroy a node in the OSM tree (fig. 1 a)

update: fill resources of a node. This can be either done by some set-value function or by directly querying the filter module.

retrieve: read resources of a node. The screen reader has to retrieve information in order to present a node.

navigate: allows movement in the tree. This can either be done by structure (parent-child / previous-next) as well as by position (screen layout) (fig. 1 b)

traverse: loop / for-each function that performs a given action/function (i.e. search). Possible processing orders are top-down and bottom-up in prefix as well as in postfix sequence.

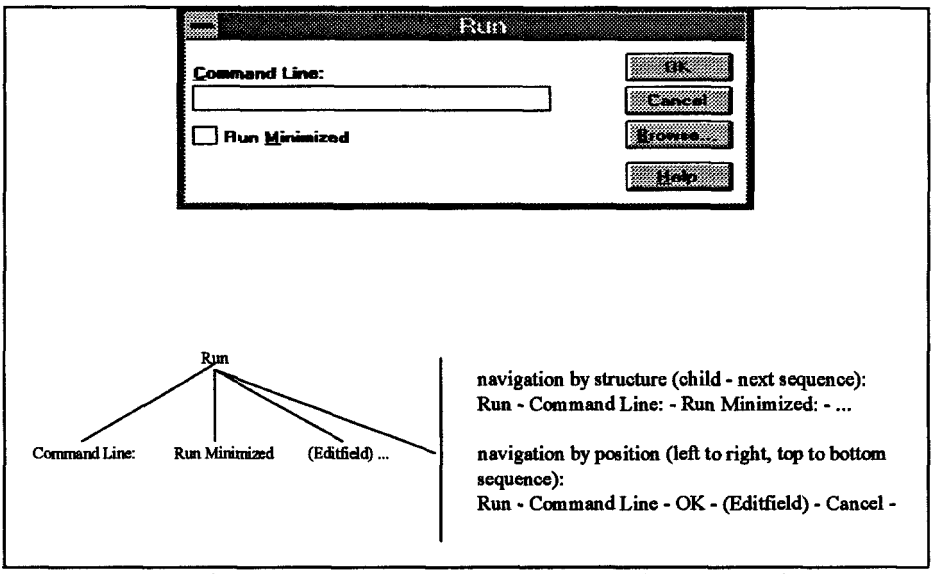

Fig. 1. a) internal structure of an OSM segment. b) example for navigation in the OSM

Different interaction objects have to be represented in the OSM. Although a common object for all possible interaction objects could be used a better solution is to have unique objects for each type of interaction objects. Different objects may have the same appearance. For example an application based on the Athena widget set will use XawButton as its button class while another build for Motif will utilise XMBut-

ton. This results in the need to map classes of interaction objects in the GUI to the class internally used in the OSM. Such a mapping should not be directly coded into the OSM source code. Instead it should be handled in a more flexible way i.e. by an alias list.

While different node types in the OSM already contain the information what object they model each node needs to hold additional information in so-called resources. Resources can be devided into three groups:

organisatorial:	an unique Id, the node type, pointers to "close" relatives, information source id
common:	position (x, y, z) and dimension of the object, its name, the widget (class) name of the object within the GUI
individual:	i.e. status of a button, formatting properties of a label or fontsize used in a textfield.

Since different classes of interaction objects most likely use different resource names a mapping mechanism similar to the one for classes is necessary.

Filter module

In order to have a uniform interface between the OSM and its sources of information all requests and their corresponding replies should be bundled in a separate module. This separation will make the OSM independent from changes in the filters as long as the quality of information remains the same. The filter module is also responsible to pass events that possibly cause changes in the OSM to it or the screen reader.

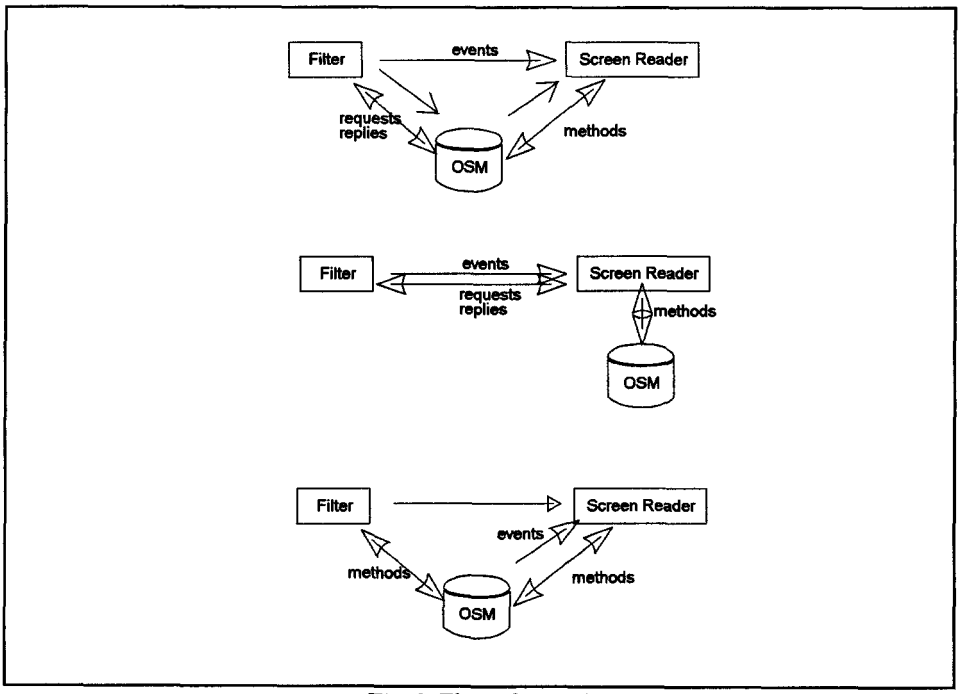

Fig. 2. Flow of control

Fig. 2 shows the flow of control in different OSMs and screen reader systems using filters. In a) the screen reader is triggered by events coming either directly from the filter module or through the OSM making the separate connection between filter and screen reader obsolete. Depending on internal rules update methods of the OSM are called. The OSM will then request necessary information from the filter and fill resources with the replies. Upon successful update the screen reader can then present the changes according to the chosen presentation model.

In b) the update will be more tightly controlled by the screen reader. Requests to the filter are issued by the screen reader. The resources in the OSM are filled with information extracted from the replies by using set-value calls. Although this gives a better control over what is updated it contradicts the separation between the screen reader and the internals of the GUI. This separation is broken because the screen reader has to deal with actions such as class mapping that should be a clear task of the OSM.

When a modified toolkit is the source of information the flow of control follows the one shown in c). Every change in the state of a filtered application is controlled by functions of the toolkit. The toolkit (filter) can therefore keep the OSM updated at every moment. From there on following the flow of control like in a) the screen reader will ask the OSM to update interaction objects. Since the OSM is always up-to-date this update will immediately report success.

Models of presentation

Depending on the quality of information available to the screen reader through the OSM and the output medium/media being used different presentation models can be used.

In case the OSM contains only lexical information a list-based presentation is most likely to be used. A hierarchical or spatial presentation is possible if lexical as well as syntactic information is available.

List-based or hierarchical models can be used with either speech or braille while the spatial model is only suitable for braille or a combination of speech and multi-dimensional sound or a combination thereof.

Conclusion

The proposed architecture has proven to be suitable and useful to give blind users access to three different platforms and GUIs. A working system for MS-Windows and X_windows has been developed in project GUIB. Future work is investigating the architecture of filters in NextStep. The increasing awareness by developers of GUIs of users with special needs will hopefully result in better integration of filters. Standardisation of interfaces in future releases or new developments of their products may improve the completness of an OSM.

Acknowledgements

This work has been supported by the GUIB consortium, a pilot research project of the EEC program TIDE (Technology Initiative for the Disabled and Elderly). I would like to thank my colleagues within the consortium as well as A. Vogel and M. Brandner for their work in X-Windows and NextStep.

Literature

[1] Boyd, L.H., Boyd, W.L., and Vanderheiden, G.C. (1990): 'The graphical user interface: crisis, danger, and opportunity', Journal of Visual Impairment and Blindness, 84, 12, pp. 496-502.

[2] Gill, J. (1993): Acces to Graphical User Interfaces by Blind People, RNIB, London, ISBN 1-85878-004-7

[3] Schwerdtfeger, R.S.: Making the GUI talk. BYTE, Dec. 1991, pp.118-128.

[4] Gunzenhäuser, R. and Weber G. (1994): Graphical User Interfaces for Blind People, in: Proceedings of 13th World Computer Congres, Hamburg August 28 - September 2, 1994, Elsevier,Amsterdam, in print

[5] Harness, S., Pugh, K., Sherkat, N., Whitrow, R. (1993): Fast Icon and Character Recognition for Universal Access to WIMP Interfaces for the Blind and Partially Sighted, in Ballabio, E.; Placencia-Porrero, I.; Puig de la Bellacasa, R. (eds.) Rehabilitation Technology, IOS Press, Amsterdam, pp. 19 - 23.