ICTA'07, April 12-14, Hammamet, Tunisia

Model Driven Approach For the Design and the Development of Non Visual Components

Amina Bouraoui , Majdi soufi

Research Unit of Technologies of Information and Communication UTIC
5, Av. Taha Hussein, B.P. 56, Bab Menara, 1008 Tunis, Tunisia
hannibal.a@topnet.tn; soufi.majdi@gmail.com

Abstract

This paper describes reusable components that can be used by developers to implement applications for visually handicapped users. The non visual components are widgets adapted from graphical ones and they use specific input/output devices and specific interaction principles. They are implemented as components to facilitate the component based development of non visual applications. The contribution of this work in the field of assistive technology is valuable, because there are no existing tools that facilitate the creation of interfaces for the blind users, and it may considerably improve computer access for this category of users.

1. Introduction

Non visual applications represent a specific type of software development. Creating software for people with visual deficiencies can be achieved using a small number of options : adapting existing software, using accessibility features of existing environments, or developing specific interfaces and applications using specific peripherals and interactions. A software developer confronted to the task of creating an application for blind users, has to face the use of complex peripherals, and different interaction principles like multimodality. The existing integrated developing environments (IDE) can be extended to deal with these specific implementations [1]. In order to create easily non visual applications, existing IDEs have to integrate the following modules : libraries to hold communication with specific I/O devices, functions to deal with I/O events, functions that implement the different I/O modalities, and a set of non visual controls to facilitate the development of adapted representation of information and interaction.
For instance a GUI developer, has a set of widgets he/she manipulates visually, by "drag and drop", giving them the position, shape, and content they will have at runtime. These widgets are the main components of an interface. It would be interesting if a non visual application developer has a matching set of controls that can be represented on adequate I/O devices.
Some attempts have been made by the past to create non visual controls, these non visual widgets used only sound modality [2, 3], and/or cannot be used to facilitate the development of interfaces for the blind users. Some attempts failed because of the immature technology. A lot of projects [4] existed in this field but no tools were created to help developing non visual interfaces and interaction objects.
The objective of this work is to help creating stand alone applications and to facilitate the development of specific applications for blind people. The advantage is that no screen reader is needed, and that the proposed widgets can be integrated in any existing IDE. Accessibility features are built inside of them and they can be used for both purposes: classic applications and non visual applications. This work is done in the context of a project aiming at the development of tools to promote the creation of specific applications for the blind and visually impaired people. This work is done essentially by extending existing frameworks [1]. Our objective is not to adapt existing software but to develop new software using the "design for all" principles [4]. The following paragraphs present the specification, the design, the implementation and the evaluation of the non visual components. All these sections are illustrated with examples in order to facilitate the comprehension for the newbie reader in the field of assistive technology.

2. Specification

Let's consider an example: some choices are to be given to the visually impaired user, and the output device is a Braille display coupled with a speech synthesizer. The developer picks up a control in the component gallery provided by the authoring tool. This

widget is called NonVisualMenu. The developer decides for its general properties: name, items, interaction devices…etc.
The developer enters the different items of the menu and attaches them to the possible events given for the control : interactive key click followed by enter, or interactive key double-click, or standard keyboard event. Then he/she defines the possible functionality (callback) for each menu item. In the final application, the component manages with the presentation of the menu on the chosen device, receives, and interprets events, and calls the adequate functions that deal with the recognised events. Fig. 1 is the original menu, and Fig. 2 is its Braille and speech adaptation using NonVisualMenu control : The lower part of Fig. 2 shows four keys that allow the user to navigate in the menu. Four of the Braille terminal function keys can be used and are programmed to navigate in the menu: up, down, next group, previous group. If the Braille terminal does not have function keys, it is possible to use the standard keyboard.

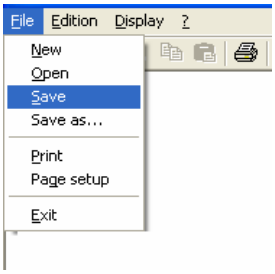

Figure 1 : An example of menu

The “UP” function key of the Braille terminal displays the menu item “Open”, the speech synthesis system says “Open”.
The “DOWN” function key displays the item “Save as”, the speech synthesis system says “Save as”.
The “PAGE UP” function displays the menu item “File”, the speech synthesis system says “File”.
The “PAGE DOWN” function key displays the item “Print”, the speech synthesis system says “Print”.
When the user presses an interactive key followed by a Braille <ENTER> (resp. double clicks an interactive key) the displayed menu item is activated and the appropriate function is called.
The same menu can be displayed horizontally with a separating character between items, function keys are adapted consequently.
In the same way we studied the example of a menu, other controls have been adapted and implemented that offer an equivalent for a developer of non visual applications, as they exist for GUI developers.

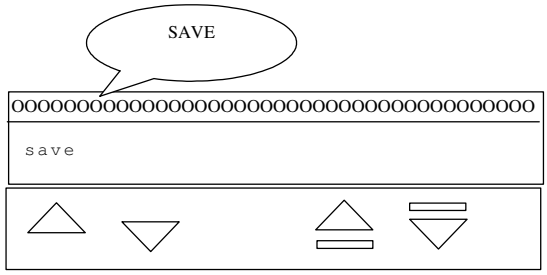

Figure 2 : Braille and speech adaptation of the menu.

The proposed components are the following : edit box, list box, menu, button. They can be integrated in composite controls such as dialog boxes, or windows.
The creation of a non visual application using these controls, creates at the same time a matching graphical but very simple application. This is necessary for visual feedback when still developing, testing or maintaining the application. It’s also useful to create dual interfaces, used by both visually impaired and sighted users [5].

3. Architecture

To encourage abstraction, the controls proposed respect a four-level architecture. The control level, the non visual control level, the functionality level, and the physical device level (Fig.3). All these levels respect defined models.

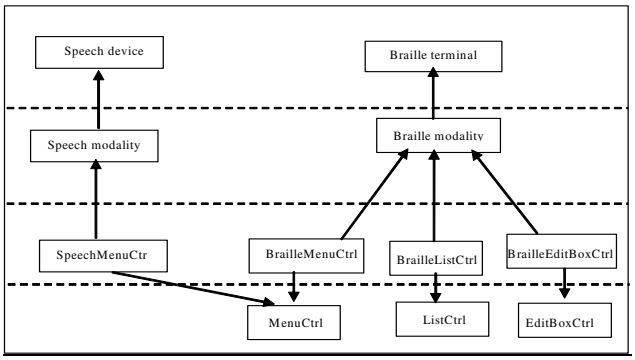

Figure 3 : An instance of non visual control architecture.

3.1 The control level

This level consists in the graphical interface provided to the developer. For the menu example this level contains the graphical tools that allow the developer the creation of the menu, its items and its basic features : title, aspect, modalities…etc. This component is the same whether the developer chooses to implement a Braille control, a sound control, a speech control or a multimodal control.

3.2 The non visual control level

It is composed by the behaviour of the control on the chosen devices. For example, how will the menu behave on a Braille display, what will be displayed first, how the items are displayed since the principal menu, when are the items read by a speech synthesizer, etc.

3.3 The functionality level

This level is composed by the functions that implement the different modalities of a control. The Braille modality, or speech modality are examples for this level. This level represent all the capacities of a specific device, and will be instantiated according to each specific device model.

3.4 The device level

It defines models of I/O peripherals that can be used, and how communication is established between them and the application, and how events are received, interpreted, and produced [6].

Eurobraille is an example of Braille terminal device, and Microsoft Speech API is an example of speech synthesis (in this case the device is software).

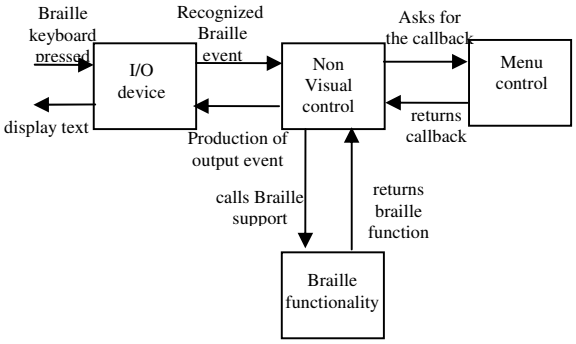

Figure 4 : Managing an input event for a control.

Fig. 4 shows the interaction between the four control levels in the case of a menu control represented on a Braille terminal. The event is received by the lower physical level -the Braille terminal-, is interpreted and then sent to the non visual control level. The non visual control asks the menu control for the associated callback when receiving this event, the menu control returns the action to do. The non visual control asks then for a Braille support from the functionality level and produces the output event.

4. Design of the components

We used the model driven approach to develop the proposed components. MDA[7] is an approach for application specification and interoperability which is based on the separation of concerns between domain knowledge and platform specificities. It relies on the use of MOF (Meta Object Facility) meta-models and UML (Unified Modelling Language) models for any step of the application life cycle, since the expression of constraints, design and analysis, to the generation of code.

This approach depends on the definition of :

- a specification model called Computational Independent Model or CIM,
- a conception model called Platform Independent Model or PIM,
- an implementation model called Platform Specific Model or PSM,
- a set of model transformations.

The control components are designed using the UML language. MOF (Meta Object Facility) formalism is used to define the component meta model of the components.

The advantage of this approach is the permanence of the conception models. If there is a need to add a modality, to change the device for a component, or if a new technology appears, the models are modified, and the code is generated. Another advantage is that the conception models become productive.

At present, we have adopted this methodology in our works, and we used Objecteering/UML environment to support the model driven approach.

We have specified the MOF meta-model of the components, and the UML platform independent models corresponding to the four levels of the non visual controls.

The source code is generated partly in C++. However, we have to make some enhancements to the mapping models in order to generate the complete components source code.

5. Implementation

The proposed components must be platform independent, so the developer is not obliged to use a unique programming workbench.

The non visual widgets can be used since C++ or Basic or Java languages, in a Windows environment. from .net or php or other platforms. For these reasons we proposed the Microsoft COM technology to implement them. This technology (integrating OLE, Active X, COM and DCOM) creates object components that offer one or more interfaces that can be called from different languages. A COM client application, can use one or more COM objects, as if they were part of it. The COM components have a unique identifier, and are used via their exposed interface. MDA is not yet mature for the generation of components source code.

There is a lot of work to do to create transformation tools adapted to our needs.

5.1 The component architecture

The partial source code generated by Objecteering/UML starting from the specified UML models, has been intensely modified using MSVC++ IDE. Other IDE's can be used if they are more adapted to the task, for example we can use Visual Basic to implement a sound functionality component.

A COM component has the following features:

- current methods and events : they are inherited from the mother class of a control, for example CEdit class;
- user methods and events : they are newly defined for the reusable control, for example non standard events;
- a graphic aspect : this is not mandatory and in that case the component is assimilated to a library (a set of functions).

The public methods and events are the exposed interface of the COM component: the way that allows the developer or other applications to use it.

The controls can respond at the same time to user defined events (Braille, speech recognition…) and standard events (keyboard, mouse…). This facilitates the creation of a non visual application that can also be used by a sighted user. Depending on the fact that they have or not a graphic aspect, the controls can either be reusable controls or reusable libraries.

The four levels of our proposed architecture are implemented as independent COM components. These components communicate using their particular interfaces. The independence of the levels authorizes the reuse of components for other applications.

5.2 The non visual widgets implementation

The device components, which manage the input and the output events of assistive devices, are implemented as ActiveX dynamic link libraries (DLL), and so are the functionality components, which define the Braille modality and the speech modality.

The non visual widgets are implemented as OCX's which are components written in C++ language.

At present, the developed components are an edit box, a list box, a button and a menu. They are respectively called EditBoxNV, ListBoxNV, ButtonNV and MenuNV. The most representative component is the menu, that we are going to describe in this paragraph. Fig. 5 shows the non visual menu inserted into a Visual C++ project.

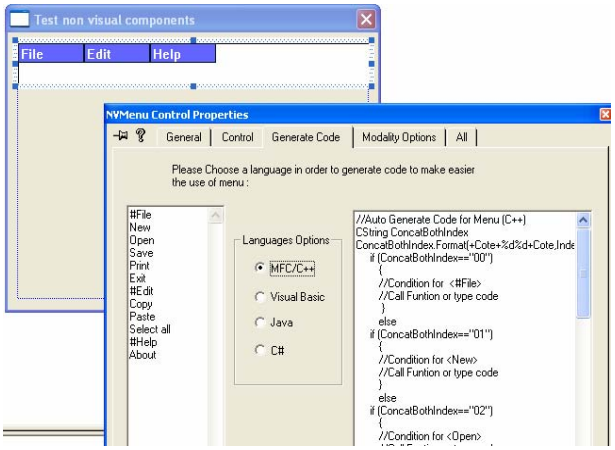

Figure 5 : MenuNV inserted into a MSVC++ project.

Fig. 6 shows an example of a non visual menu into a VB container application at runtime, and the focus is on "File" menu. For instance this menu is represented horizontally. The window on the right of the figure is an application that simulates what is displayed on the Braille terminal. This is useful for the developer in order to have a visual feedback. The developer doesn't necessarily understand the Braille language.

Figure 6 : Braille simulation of the menu.

6. Sample application

In this section we present a simple application developed with our components, and we discuss the possibility to use the components with Java platform.

6.1. A Web application

This example shows the use of the non visual components into a simple dynamic website developed with PHP and MySQL (Fig. 7). This website gives the user the opportunity to search and to buy a book. The components used are the button and the edit box.

The user can move through the different components using the tabulation key. Other keys allow the user to have extra information.

Fig. 8 shows how to configure the ButtonNV control.

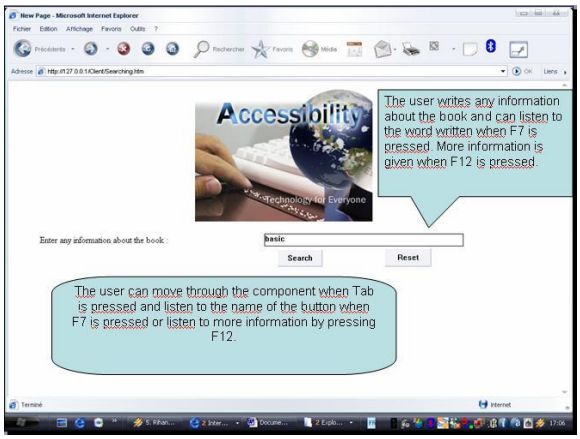

Figure 7 : The start up page of the website

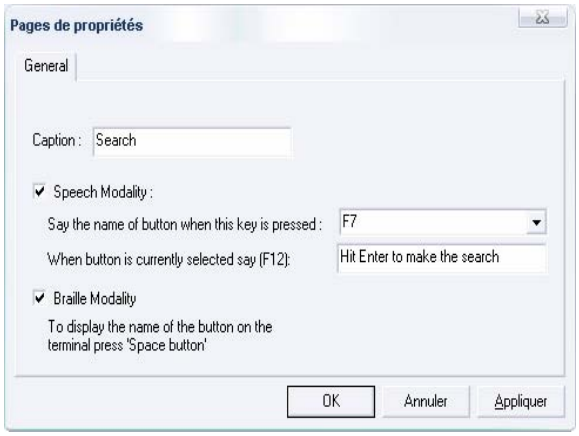

Figure 8 : Setup for ButtonNV

When the user launches the search, the results page is displayed (Fig. 9).
The results page uses several times the non visual edit box and button. The user listens to page contents when s/he moves trough the different controls. The Braille display is updated with each control change.

6.2 Non visual components and Java Platform

Using the components from Java platform is possible but more difficult because COM components can not be used without being converted to JavaBeans. There are tools that can help the developer to do this conversion, such as the Migration Assistant, or Bridge2Java which help to convert OCX files to Java files.

7. Evaluation

The evaluation process has been made at two levels : the use of the components for a developing purpose by a developer, and the use of the components in a container application by a blind user.

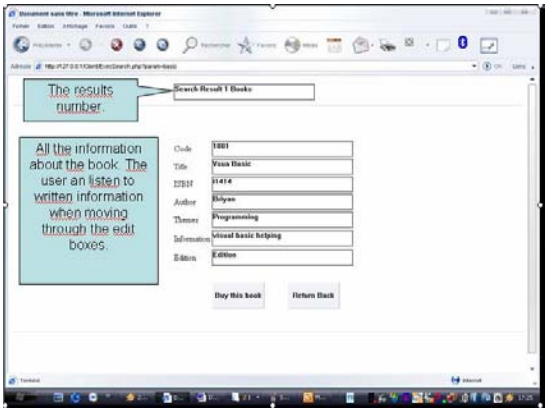

Figure 9 : The website results page.

7.1 The developer mode

Developing tests have been made with Visual Basic IDE, MSVC++ IDE, and .NET platform.
Developers in our research unit, which have not been involved in the components project, have used them for implementation purposes in Visual Basic IDE. They spent a short time in understanding their functionalities and integrating them into an application. Their feedback is that they found no differences in use between the non visual components and other OCX and VBX components they used before, besides the fact that they found the components very intuitive to use, and well documented.

7.2 The user mode

We have used the non visual components to develop simple applications in different environments, in order to test their portability and reusability. For instance :

- a simple browser was developed using php platform, allowing the search of books on the Internet;
- a word processor was developed in Visual Basic, the application has basic features such as text entering, saving, and printing. Fig. 10 shows the word processor application. The upper part of the application main window contains the non visual menu component MenuNV. The toolbar in the middle of the window has been built using several times the component ButtonNV. The lower part of the screen contains the two other components : ListBoxNV on the left, and EditBoxNV on the right. The toolbar represents the most used functionalities of the application. The list box displays the most recently opened files, and the edit box displays the file content. The speech synthesis states the name of the component having the focus, while its content is written on the Braille display (that is simulated on the second window of Fig. 10).

The implementation took a very short time, and the application was given to a blind user without further explanations. He found it usable, and very friendly as

it was faster than what he was accustomed to. (He usually use a screen reader under Microsoft environment). The user has also proposed some enhancements we are planning to integrate to our developments, such as multilingual speech synthesis. Actually our OCX's work in French and English languages but can easily be adapted to other languages.

8. Conclusion and further work

The developed widgets are used to extend a programming environment [1,8], and can be easily used by developers, the same way they use other controls. The four level architecture is composed by reusable components that can be used independently and for different purposes.

However, this work can be improved if we apply completely the OMG's Model Driven Architecture [7]. Our future work consists mainly in applying the OMG's MDA approach explained above to create the non visual controls. The advantages of this approach are possibility to create controls for various platforms starting from the same specification, and the fact that the emergence of a new technology or platform affects only the Platform Specific Models.

Besides the application of the MDA approach, we want to integrate more I/O devices for the controls, such as the speech recognition as an input device; define and implement more controls; define new controls that are not adapted from existing controls. We are also planning to apply multilingual functionalities to speech synthesis and to adapt Arabic language to the Braille modality.

Another perspective is now under study, to create components with zoom, contrast and colour enhancements for the visually impaired users. These components will be used for magnification of information. They are aimed to respond to the visually impaired users requirements in terms of magnification ratio, contrast, use of colours, etc.

At the end of this work we will be in presence of a reusable control repository that allows developing rapidly applications for different categories of visually handicapped users.

The main advantage of our work is the developing of specific applications for the blind users very rapidly and at a low cost. For instance, these controls are valuable in the education field for blind children in Tunisia. By installing applications containing non visual components on each computer in a classroom, we can imagine the gain compared to the use of a screen reader at a prohibitive price.

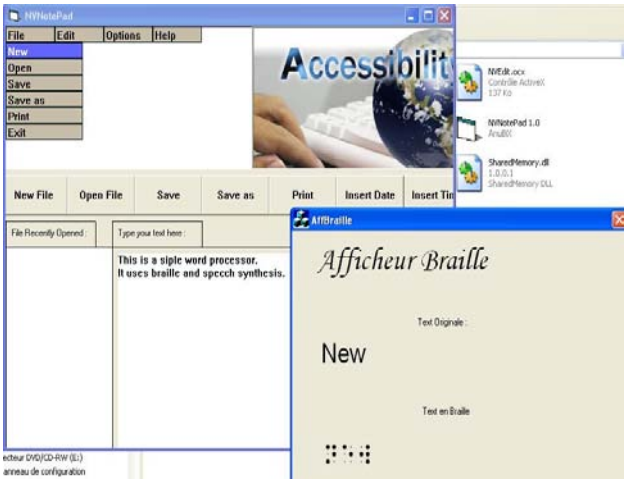

Figure 10 : A component based word processor.

9. References

[1] Bouraoui A., Extending an existing tool to create non visual interfaces. *Proceedings of CSUN 2006, Center on Disabilities 21st Annual International Technology and Persons with Disabilities Conference* , California State University Northridge, 2006.

[2] Brewster S, The Design of Sonically-Enhanced Widgets, *Interacting with computers*, *11* (2), 1998, 211-235.

[3] Edwards A.D.N, Mitsopoulos E.N, A Principled Methodology for the Specification and Design of Non-Visual Widgets, *ACM Transactions on Applied Perception*, *2*(4), 2005, 442-449.

[4] Emiliani P.L, Stephanidis C, From Adaptations to User Interfaces for All, *Proceedings. The 6th ERCIM workshop on "User interfaces for all"*, Italy, 2000.

[5] Savidis, A., Stephanidis, C., The HOMER UIMS for Dual User Interface Development: Fusing Visual and Non-Visual Interactions. *Interacting with Computers, 11* (2) 1998, 173-209.

[6] Wang A.J.A, Diaz Herrera, J.L, Device drivers as reusable components, *Proceedings of Software Engineering and Applications*, Marina Del Rey, USA, 2003.

[7] OMG Model Driven Architecture resource page. Needham, MA: Object Management Group. Internet : http://www.omg.org/mda/index.htm. 2003.

[8] Stephanidis, C, Emiliani , P..L , Universal Access to Information Society Technologies: Opportunities for People with Disabilities, *Proceedings of Computer Helping People with Special Needs : 8th International Conference*, Linz, Austria, 2002.