

Caring, Sharing Widgets: A Toolkit of Sensitive Widgets

Murray Crease, Stephen Brewster & Philip Gray

Department of Computing Science

University of Glasgow

Glasgow, UK, G12 8QQ

Tel : +44 (0)141 339 8855, +44 (0)141 330 4966

Fax : +44 (0) 141 330 4913

Email : <murray,stephen,pdg>@dcs.gla.ac.uk

Web : <http://www.dcs.gla.ac.uk/~murray,~stephen,~pdg>

Although most of us communicate using multiple sensory modalities in our lives, and many of our computers are similarly capable of multi-modal interaction, most human-computer interaction is predominantly in the visual mode. This paper describes a toolkit of widgets that are capable of presenting themselves in multiple modalities, but further are capable of adapting their presentation to suit the contexts and environments in which they are used. This is of increasing importance as the use of mobile devices becomes ubiquitous.

Keywords: audio, multi-modal, resource-sensitive, sonically enhanced widgets, toolkit

1 Introduction

Many modern applications are designed to run on powerful workstations with large monitors and powerful graphical capabilities. Whilst this is often the case, perhaps the user wishes to run the application on a less powerful laptop, or perhaps even on a hand-held device. In these cases, the application will not have access to the same amount of visual output resource available to display its interface. Even if it is the case that the application is running on a powerful workstation with a large monitor, the feedback provided by the graphical interface may be better provided by a different sensory modality or a combination of several modalities. Or perhaps the application is running on a mobile telephone where graphical feedback is only suitable when the user is not making a call. These examples highlight the need for interface objects, or widgets, which are capable of adapting their presentation according to different requirements: the suitability of different presentation resources, e.g. in a loud environment the use of audio feedback may be unsuitable and the availability of different presentation resources, e.g. the resolution of the screen being used or the number of MIDI channels available. For the widgets to be

able to cope with these different demands, they need to be able produce many different forms of output. Whilst it would be possible to build the widgets with built in output in many different output modalities, this would limit the possibilities. It would be better to be able to easily change the widget's output to allow the evaluation and inclusion of new forms. This paper describes a toolkit of such widgets and goes on to discuss some of the issues which arise from the implementation of the toolkit.

2 The Aims Of The Toolkit Of Resource Sensitive Widgets

The toolkit described in this paper has several aims. Its widgets should be multi-modal, with no one modality assumed to be of any greater importance than any other. We use the phrase modality to refer to a sensory modality so, for example, all auditory output is one sensory modality and all visual output is another modality. The widgets should be sensitive to their environment, adjusting their feedback accordingly. It should be easy for other user interface designers to change the feedback, either personalising the existing feedback or replacing it with a completely new design. Finally, at all times the feedback given to the user should be consistent, regardless of the modality used. That is, regardless of modality, the information given to the user by a widget should be the same.

Most modern human computer interfaces are based around graphical widgets. This is despite the fact that most of us in our everyday lives use all our different senses to interact with the rest of the world and that many modern computers are capable of generating feedback in modalities other than vision (most notably sound, but in some instances, touch). To rectify this situation, the toolkit's widgets are designed to be multi-modal, with every modality treated equally. Our previous work (Brewster, 1998) has shown that the addition of audio feedback can improve the usability of an interface as has work done by (Gaver, 1989) whilst haptic feedback has also been shown to be effective (Akamatsu & Sato, 1994, Oakley *et al.*, 2000). In these cases, however, the additional feedback was supplemental to the graphical feedback as opposed to being an equal partner. Our toolkit's widgets avoid the assumption that visual feedback is of greater importance than others, allowing for the possibility of widgets which are, for example, audio or haptic only. This is important in situations where the use of visual feedback is limited, e.g. on a mobile device where screen space is limited; impossible, e.g. on a mobile phone when the user has the phone to his/her ear; or unsuitable, e.g. if the user is visually impaired. If the widget is capable of utilising many modalities and makes no assumptions about what modality is best, these situations can be dealt with.

If a widget is capable of utilising multiple modalities equally, it becomes easy to imagine situations where it would be desirable for the widget to switch between modalities. For example, it has been shown that audio feedback can be just as effective as visual feedback in conveying information about the progress of a

background task (Crease & Brewster, 1999), and that audio feedback can compensate for a reduction in the size of graphical buttons (Brewster, 1999). This switch may be made by the user to personalise the widget's feedback, or by the system, either to reduce the load on an over-stretched resource or to utilise a more suitable modality for the given context of use. A user may wish to alter the usage of a modality through preference, he/she thinks the sounds are too loud and would like to make them quieter; or through necessity, e.g. an hearing-impaired user has no need for audio feedback. These changes may merely alter the balance of usage for the different modalities, e.g. a little more audio feedback and a little less visual feedback, or may remove a modality all together. The system may change the modalities used by a widget for two reasons. If, for example, there are insufficient resources to meet the demands of a widget in a particular modality the system will reduce the amount of resource required by that modality and may attempt to compensate by increasing the utilisation of a different modality. The other scenario is that the system recognises that the current utilisation of a particular modality is inappropriate in the particular context and will amend the usage appropriately. For example, if the ambient volume in the environment increases, the system may increase the level of the audio feedback given to compensate, or if that is inappropriate, it may switch to a different modality entirely.

When widgets are capable of utilising multiple modalities, the job of designing the widget's feedback becomes harder. Standard graphical widgets are well established and, rightly or wrongly, are almost a *de facto* standard. Audio feedback, for example, is less well established and there are many conflicting designs for the feedback. For this reason it is important that new designs for the presentation of widgets can be easily included. These designs may replace the existing designs in a particular modality or may supplement the presentation by using other modalities. Modifying or replacing the design of feedback in one modality should not affect the feedback in a different modality if the feedback produced in each modality is independent. For example, if the toolkit currently used audio feedback based around structured, non speech sounds called earcons (Blattner *et al.*, 1989, Brewster *et al.*, 1993) but a designer wished to include an alternative design for audio feedback using everyday sounds representing the events taking place called auditory icons (Gaver, 1986), it should be easy it replace the earcons with auditory icons without affecting the existing visual feedback of the widgets. Additionally, making the introduction of new designs easy, enables designers to incorporate their designs into existing applications decreasing the overhead of evaluating new designs.

Regardless of which modalities are used, it is important that the feedback generated is consistent, both between widgets and within a widget between modalities. If a widget switches between modalities, the overall feedback must give consistent information. If a widget utilises two or more modalities the information given in each must be consistent. For example, if a progress indicator is visible on the screen, but is then covered by a different window, the audio feedback must give information that is consistent with the graphical feedback given previously. Similarly, different widgets must be presented consistently throughout the system. For example, two different buttons may have different audio styles, e.g. one may use a pop style and one may use a jazz style, but they both must present their information in a way that the user recognises as having the same meaning. To this end, they may use different timbres to achieve their styles, but the same rhythm to

maintain consistency. Additionally, clashes between the feedback given by different widgets must be avoided. This issue is usually resolved for graphical feedback by assigning different graphical widgets to different areas in the 2D region which makes up the visual display on a screen. Further, this 2D area simulates a third dimension by allowing the areas occupied by widgets to overlap. For other forms of feedback, the problem is harder to resolve. Two pieces of feedback presented at the same time will attempt to occupy the same “space” in the output region, potentially causing a clash. It is up to the system to avoid such clashes by limiting the amount of feedback given in such a modality or by somehow modifying the feedback so it no longer clashes, e.g. by ensuring two sounds employ different timbres to aid their distinction, or perhaps using a different modality for one of the pieces of feedback.

2.1 Two Examples

Here are two possible scenarios that illustrate the need for our toolkit:

Murray is running an application on a mobile device. He is sitting on a train in a quiet carriage. The application’s widgets can only use a limited amount of screen space due to the limited size of the device’s screen. To compensate for this, the system utilises audio feedback to supplement the visual feedback. Because the carriage is quiet, the audio feedback is at a low level. At the next station, a family with young children enters the carriage, increasing the ambient volume level. To compensate for this, the system increases the level of the audio feedback. At the next station, Murray gets out and walks along a quiet street in bright sunlight. To compensate for the walking motion, the system increases the level of the audio feedback as it is unable to increase the size of the widgets due to the limited screen size. To compensate for the sun shining on the screen, the system increases the contrast of the screen.

When Murray gets home, he switches to using his desktop machine which boasts a large monitor and is attached to an external sound synthesiser. The system is able to utilise a lot of visual feedback because of the large screen, but still uses audio feedback because Murray likes it as a supplement to the visual feedback and the system is capable of producing high quality audio output. The audio feedback is generated using the synthesiser. Murray runs a MIDI sequencer application and starts to compose a tune. When the tune is played, this increases the ambient volume in the room so the system attempts to increase the level of audio feedback to compensate. However, because most of the MIDI channels are used playing the tune, the system is unable to meet this demand. To handle this lack of resource the level of audio feedback generated by the application is decreased and to compensate the graphical feedback is increased.

3 The Toolkit Design

The previous section described the four main requirements to be met by the toolkit. Each widget should be capable of producing feedback in multiple modalities with no preference given to any particular modality. The widgets should be capable of using the modality most suitable or limiting the use of a modality which has limited resources. It should be easy to change the feedback a widget produces in one or more modality with no effect on any other modalities. The feedback given should be consistent, both between widgets and between modalities. This section describes how the design of the toolkit meets these four aims.

3.1 The Toolkit Architecture

Figure 1 shows the architecture of the toolkit. The widget behaviour accepts external events and translates them into requests for feedback. These requests do not specify the modality to be used for the feedback, but rather just the meaning of the feedback, e.g. the mouse is over the widget. By accepting only the events that are relevant to the widget in its current state, the behaviour of the widget is defined.

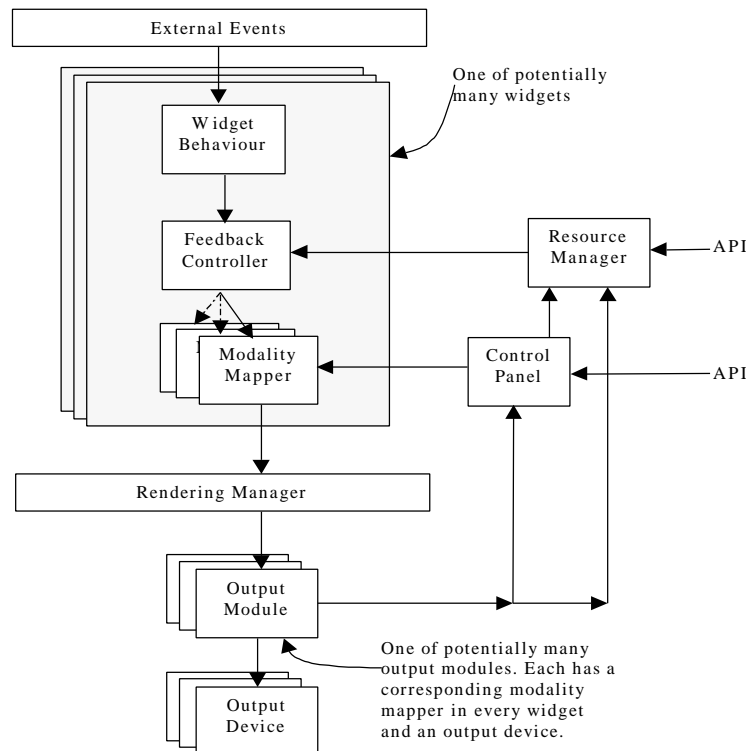


Figure 1 – Toolkit architecture.

The requirement that the widgets are multi-modal and treat each modality equally is met by the feedback controller. Because the widget only requests the feedback in terms of semantics, i.e. the mouse is over the widget, the requests can be translated into multiple forms of feedback in different modalities. This is done by the feedback controller which splits the request made by the widget behaviour into suitable requests for feedback in potentially multiple modalities. These requests are given a weight specifying the level or importance for that particular modality. This weight could map to the volume of audio feedback or size in visual feedback, for example. These requests are then passed on to the rest of the toolkit and ultimately result in appropriate feedback being produced.

The requirement that the widgets are sensitive to the available resources and their suitability for the current environment is met by the resource manager. The resource manager receives input from three sources, a control panel, the output modules and from applications that use the toolkit via its API. The control panel allows the users of the system to set the weight for a particular modality. The weight of a modality encompasses two concepts: the level of user preference for a modality, and the level of system resource required to meet requests in a modality. This way, users can set their preference for a particular modality. The output modules can indicate that they have insufficient resource to meet feedback requests to the resource manager, allowing the widgets to be sensitive to the availability of resources. External applications can use the resource manager's API to influence the weight of different modalities. Such applications could monitor the environment allowing the widgets to be sensitive to resource suitability. One such example of this would be an application that monitored the ambient volume level of the environment, changing the weight of the audio feedback so that it can be heard without being too loud.. The resource manager takes the information from these three sources and passes on a weight to the feedback controller for each of the modalities.

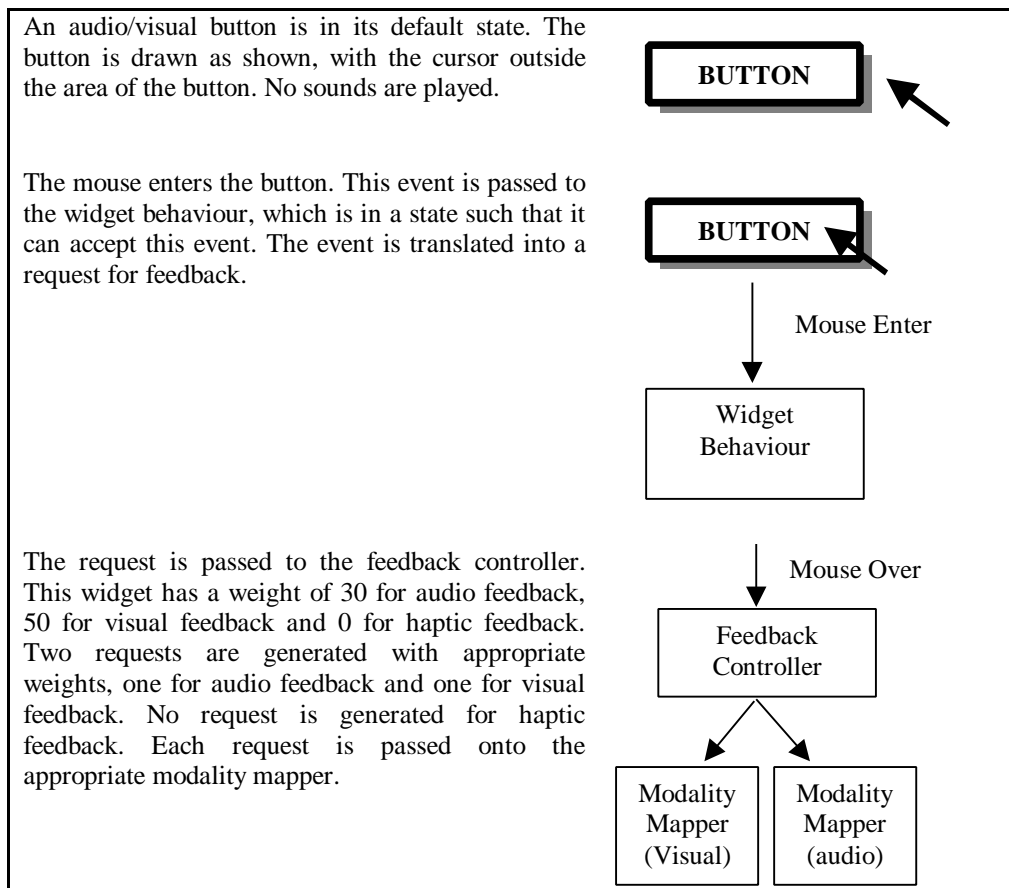
The requirement that the widget's feedback is easy to change is met by the output modules and the control panel. Because the widget behaviour does not encapsulate the feedback given by the widget, but merely requests that feedback be given, it is a simple matter to change the feedback of the widgets. To replace one form of auditory feedback with another, for example, it is simply a case of replacing the existing output module with a new one. To supplement the existing feedback with feedback in a different modality, simply add a new output module to the toolkit. The control panel enables the changing of the output of a widget according to user preference. Any options set in the control panel for a widget are added to the request made for feedback in the modality mapper. There is a modality mapper for each output module the widget uses. The parameters that can be set for each output module are supplied by that output module. For example, a visual output module may supply parameters like colour, shape, 3D and an audio module may supply parameters like pitch, jazz, rock, pop.

The requirement that all the feedback for the widgets is consistent both between modalities and between widgets is met by the rendering manager. Again, because the widget's feedback is not encapsulated within the widget, it is possible to alter a widget's feedback so it does not clash with the feedback from other widgets. An example of feedback clashing would be two similar, sounds being played at the

same time. The two sounds could interfere with each other, rendering the information they are conveying unintelligible. Because the rendering manager has a global perspective on what feedback is being requested at any given time, it can avoid such clashes by, in this case, perhaps changing the timbres of one of the sounds being played to make them more distinguishable. Another potential inconsistency is the use of two modalities which do not make sense when used together. For example, a button could be presented visually as being wooden, but the audio feedback may present the button in a metallic way. The rendering manager could detect this clash and suggest a change in the feedback. The model of what combinations are acceptable and what are not would have to be built up using information from both the user and the output modules. For example, a user could specify the maximum number of sounds to be played at any one time or an output module could specify preferred output in a different modality.

3.2 An Example

Figure 2 shows a concrete example of how the toolkit works, using a standard button.



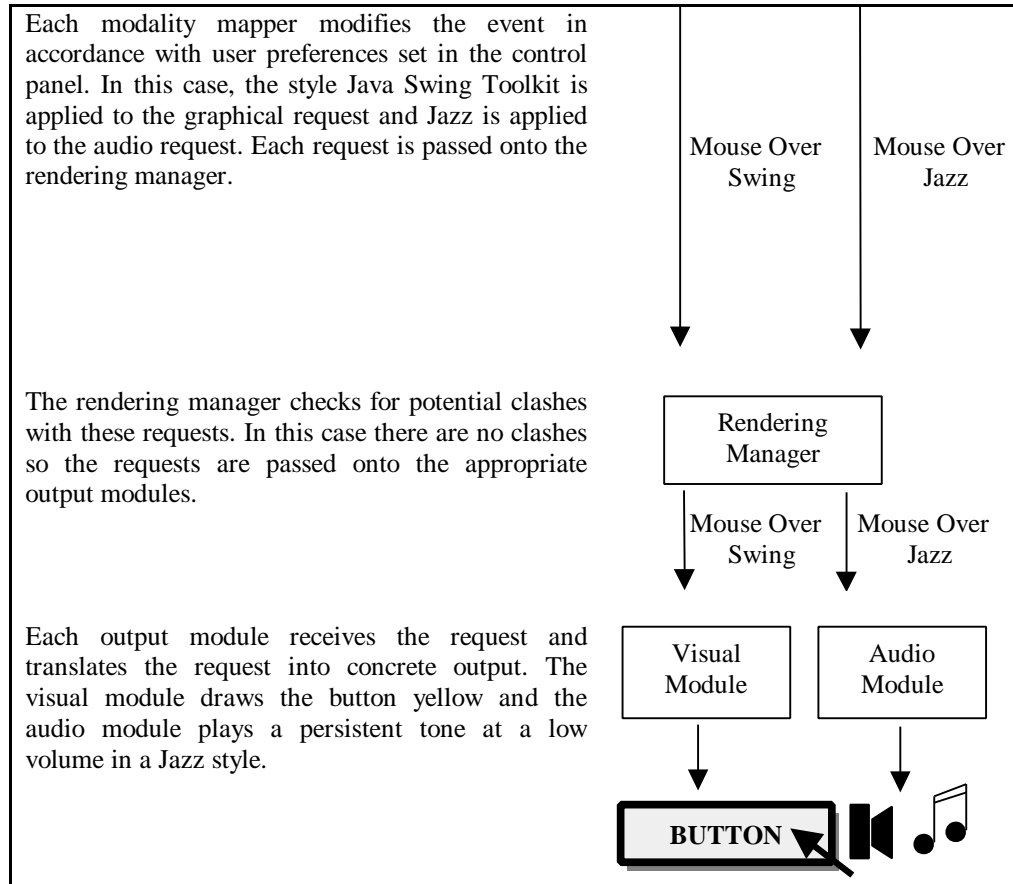


Figure 2 – An example interaction using the toolkit.

4 Related Work

The Seeheim model (Pfaff, 1985) was one of the first user interface models to separate the user interface architecture into monolithic functional blocks. Three functional blocks were defined: the presentation system which handled user input and feedback; the application interface model which defined the application from a users point of view and the dialogue control system which defined the communication between the presentation system and the application interface model. Like Seeheim, our toolkit has a monolithic presentation component (albeit only for output), although the dialogue control system is distributed through out the widgets. The toolkit does not deal with application models because it is solely concerned with the output generated by individual widgets.

MVC (Model View Controller) and PAC (Presentation, Abstraction, Control) (Coutaz, 1987) are both agent based models, where an agent is defined to have “state, possess an expertise, and is capable of initiating and reacting to events.” (Coutaz *et al.*, 1995). An interface is built using hierarchies of agents. These agents represent an object in the application. In MVC, the model describes the semantics of the object, the view provides the (normally visual) representation of the object and the controllers handles user input. In PAC, the abstraction describes the functional semantics of the object, the presentation handles the users interaction with the object, both input and output and the control handles communication between the presentation and the abstraction as well as between different PAC agents. The toolkit is object-oriented like both MVC and PAC, with each widget (or agent) encapsulated into different objects. The toolkit, however, does not define the whole user interface in terms of a hierarchy of agents, but rather defines the individual widgets without specifying their organisation. Like the MVC model the toolkit separates input and output, although unlike MVC, the toolkit’s widgets do not have a controller type object. It would be possible, however, to build an MVC type architecture around the toolkit. The toolkit’s architecture has been compared to PAC in the following way: The modality mappers and output modules are abstract and concrete presentation modules, although unlike PAC only handling output. The feedback controller and rendering manager are controllers and the widget behaviour is the abstraction (Calgary, 1999). Unlike PAC, however, the toolkit’s abstraction is only aware of the widget’s state, but is not aware of the underlying application semantics. This is because the toolkit is designed as an extension of the Java Swing toolkit, allowing it to be easily incorporated into existing Java applications.

The Garnet system (Myers *et al.*, 1990) is a set of tools which allow the creation of highly interactive graphical user interfaces, providing high level tools to generate interfaces using programming by demonstration and a constraints system to maintain consistency. The Garnet toolkit allows the graphical presentation of the toolkit’s widgets to be easily modified by changing the prototype upon which the widget is based. Doing this will update all dependent widgets. This is analogous to changing the design of output for a widget in an output module of our toolkit.

The HOMER system (Savidis & Stephanidis, 1995) allowed the development of user interfaces that were accessible to both sighted and non-sighted users concurrently. By employing abstract objects to specify the user interface design independently of any concrete presentation objects, the system was capable of generating two user interfaces which could run concurrently for the same application. This allows sighted and non-sighted users to co-operate using the same application. Unlike our toolkit, the HOMER system developed two interfaces, using two separate modalities rather than have one interface which can switch between multiple modalities as and when required, using several concurrently if appropriate.

Alty & McCartney, (1991) created a multimedia process control system that would choose the appropriate modality to present information to a user. This would allow more information to be presented by increasing the bandwidth the interface could use. Additionally, if the preferred modality is unavailable if, for example, it is already being used for output, the system would attempt to present the information using an alternative. It was found, however, to be almost impossible to specify how

these switches should be made. To limit the complexity of the system, a user-interface designer would supply it with valid options for output modalities.

The ENO system (Beaudouin-Lafon & Gaver, 1994) is an audio server which allows audio applications to incorporate audio cues. ENO manages a shared resource, audio hardware, handling requests from applications for audio feedback. This shared resource is modelled as a sound space, with requests for sounds made in terms of high level descriptions of the sound. Like ENO, our toolkit manages shared resources, although the toolkit extends the concept by switching between resources according to their suitability and availability. Similarly, the X Windows system (Scheifler & Gettys, 1986) manages a shared resource, this time a graphics server. Again, our toolkit extends this concept by managing resources in multiple modalities and switching between them.

Plasticity (Thevenin & Coutaz, 1999) is the ability of a user interface to be re-used on multiple platforms that have different capabilities. This would minimise the development time of interfaces for different platforms. For example, an interface could be specified once and then produced for both a workstation with a large screen and a mobile device with limited screen space. This is achieved by specifying the interface using an abstract model, and subsequently building the interface for each platform using that platform's available interactors and resources. Like the toolkit, plasticity allows user interfaces to adapt to available resources, although the toolkit does this at the level of individual widgets whilst plasticity does this at the level of the interface. Additionally, the toolkit attempts to adapt the interface across multiple modalities whereas plasticity is only aimed at visual feedback.

5 Current Implementation

Using Java, the framework of the toolkit has been completely implemented, allowing the widgets to have their presentation changed by both a user and the system. The API to the toolkit is complete, allowing designers to incorporate new forms of feedback into existing widgets, or to add new widgets to the existing set. Currently, the toolkit has been implemented with two widgets, a button and a progress bar; using two modalities, graphics and audio. Graphically, the buttons are Java Swing buttons and the progress bars are drawn from first principles, illustrating the flexibility of the toolkit. Swing buttons were used because this allowed us to take advantage of Swing's built in event handling system. Any changes in the weight of the visual modality are mapped directly to size.

Because the progress bar does not require complex event handling, it could be drawn from first principles. This allowed us to demonstrate the potential for more complex representations for different visual weights. Figure 3 shows the different ways visual weight has been mapped to different graphical representations in this simple example. Figure 3 (a) shows the progress bar at a low visual weight, with the progress bars in (b) and (c) having successively greater visual weights. Although

this increases the work required by the designer of the widget, in this case three visual designs are used rather than one, the work an interface designer is required to do is reduced because it is possible to use the same widget on multiple platforms taking advantage of the multiple designs encapsulated in the output module.

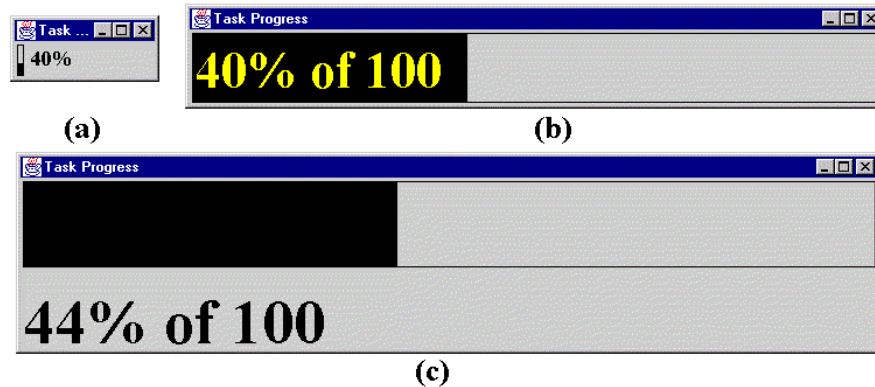


Figure 3 – Different visual representations of a progress bar for different visual weights.

The sounds used to provide the audio feedback for both the button Brewster, 1998 and the progress bar (Crease & Brewster, 1998) used earcons. Two audio output modules were developed, both of which used similar sounds, but the mappings for audio weight were different. One simply changed the volume of the sounds in proportion to the audio weight, whilst the other varied the number of sounds played. At low audio weights only the most important sounds were played whilst at higher weights more sounds were played. For example, at a low audio weight only the sound used to indicate a valid selection was played for a button, whilst at higher weights a sound indicating the mouse was over an active button was played and at even higher weights, sounds indicating that the button had been pressed were played. Using the control panel, it is possible to change the output module used dynamically.

To incorporate a resource sensitive widget into a standard Java program requires minimal changes. Figure 4 shows the code necessary to declare a standard Java Swing JButton (here called button) and add it to a JPanel (panel). Figure 5 shows the code necessary to declare a resource sensitive MButton and add it to a JPanel. As can be seen, the only changes to the code required are changing the type of object from JButton to MButton and changing the object passed to the add method of the panel. It is our intention to build a tool which will parse existing code, automatically making these changes, making the transition from standard Swing widgets even easier.

```
JButton button = new JButton("Progress");
panel.add(button);
```

Figure 4 – Adding a standard Swing button to a panel.

```
MButton button = new MButton("Progress");  
panel.add(button.getTheWidget());
```

Figure 5 – Adding a resource sensitive button to a panel.

A module was developed for the toolkit which communicated with the resource manager. This module measured the ambient volume of the surrounding environment and adjusted the weight of the audio feedback accordingly. The addition of this module to the toolkit highlights its flexibility and demonstrates how it can be made sensitive to whatever environmental factors are relevant to the user.

6 Discussion

At the moment, the toolkit has a framework to allow widgets to switch between modalities as and when this is suitable. However, further work is required to help understand whether it is possible to define how much feedback in one modality is required to compensate for a reduction in feedback in a different modality. Indeed, it seems unlikely that a generic solution which can automatically handle all possible combinations of modalities and designs could be built. A more realistic target is a semi-automatic system. This would work for specific situations, such as reducing the size of a graphical button to on the display of a mobile computer and compensating by the addition of sound. Brewster (1999) found that using sound in addition to the standard graphical representation of a button on a hand-held device allowed the buttons to be reduced in size. This demonstrates it is possible to set up the toolkit to handle individual scenarios, enabling it to compensate for the reduction in one modality by using another modality.

As can be seen from Figure 1, the toolkit's architecture is heavily biased around output, with no system controls on the input side. The widget's output is configurable, both by users and the system, whereas the input is fixed. To redress this imbalance, the input should be treated in a similar way to the output. This would entail not encapsulating the input behaviour in the widget, but rather separating it out into a different module as has been done with the output. The role of the widget behaviour would no longer include handling input events, but rather to co-ordinate input and output, providing the base upon the widget is built. In this way, new input mechanisms, for example speech or gesture input, could be designed and added to the widget with the same ease as output mechanisms. As with the output modules, this would reduce the overhead of incorporating and evaluating new input mechanisms into existing widgets.

Another issue which arises from the toolkit design regards the widgets input and output areas, e.g. the screen area used to accept input from a mouse for a widget and to present graphical output for a widget. Because the output, and potentially the input, mechanisms are no longer encapsulated within the widget it is important to

avoid inconsistencies. Further, the output area may change over time, compounding the situation. It is therefore important to ensure that the output modules communicate with the input mechanisms to maintain consistency. This dialogue would be controlled by the widget behaviour module described above. Different output modules need to communicate with different input mechanisms. An audio output module has no relevance to a mouse input mechanism. To resolve this, each input mechanism and output module could be associated with an interaction area. For example, a mouse input mechanism and a graphical output module would be associated with a screen interaction area. A spatialised sound output module and a gesture input mechanism could be associated with a 3D space interaction area in which the output is spatialised and the user then selects a target by gesturing at the area the sound is played in. This is an extension of the concept of a window in X (Scheifler & Gettys, 1986) or a sound space in ENO (Beaudouin-Lafon & Gaver, 1994), where rather than managing a single shared resource, the toolkit manages several, switching between resources when appropriate.

7 Conclusions & Future Work

This paper describes a toolkit of multi-modal, resource sensitive widgets. The toolkit's widgets are capable of presenting themselves in multiple modalities, with no preference for any one modality. If appropriate, the way a widget utilises the different modalities can be varied according to the suitability of the modality for a situation, perhaps substituting a different modality for an unsuitable one. Equally, it is simple to include new designs of feedback in an existing widget without affecting feedback in different modalities allowing the evaluation of new designs without the overhead of building the complete widget and incorporating these new widgets into an application.

Although the mechanism is in place for switching between modalities, more work needs to be done to try and understand the rules, if any, which govern these switches. With the framework it supplies, our toolkit could be a useful tool in this research.

Using the toolkit, it is now possible for designers to build interfaces that are suitable for a range of different contexts and environments. This is of increasing importance as the use of mobile devices that can be used in greatly varied environments grows.

8 Acknowledgements

This work was funded by EPSRC grant GR/L79212.

References

- Akamatsu, M. & Sato, S. (1994). A Multi-Modal Mouse with Tactile and Force Feedback. *International Journal of Human-Computer Studies*, **40**, 443-453.
- Alty, J. & McCartney, C. *Design Of A Multi-Media Presentation System For A Process Control Environment* In: Eurographics Multimedia Workshop, Session 8: Systems, (eds.), (1991)
- Beaudouin-Lafon, M. & Gaver, W.W. *ENO: Synthesizing Structured Sound Spaces* In: Proceedings of the ACM Symposium on User Interface Software and Technology, 1994, (eds.), ACM Press, Addison-Wesley, 49-57. (1994)
- Blattner, M.M., Sumikawa, D.A. & Greenberg, R.M. (1989). Earcons and Icons: Their Structure and Common Design Principles. *Human-Computer Interaction*, **4**, 11-44.
- Brewster, S. (1998). The Design Of Sonically-Enhanced Widgets. *Interacting With Computers*, **11**, 211-235.
- Brewster, S. *Sound In The Interface To A Mobile Computer* In: Proceedings of HCI International'99, (eds.), Lawrence Erlbaum Associates, NJ, 43-47. (1999)
- Brewster, S.A., Wright, P.C. & Edwards, A.D.N. *An Evaluation of Earcons for Use in Auditory Human-Computer Interfaces* In: Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems, (eds.), 222-227. (1993)
- Coutaz, J. (1987). PAC: An Object Oriented Model for Implementing User Interfaces. *ACM SIGCHI Bulletin*, **19**, 37-41.
- Coutaz, J., Nigay, L. & Salber, D. (1995). Agent-Based Architecture Modelling for Interactive Systems. *Critical Issues In User Interface Engineering*, 191-209.
- Crease, M. & Brewster, S. *Making Progress With Sounds - The Design And Evaluation Of An Audio Progress Bar* In: Proceedings Of ICAD'98, Edwards, A. & Brewster, S. (eds.), British Computer Society, (1998)
- Crease, M. & Brewster, S. *Scope For Progress: Monitoring Background Tasks With Sound* In: Human-Computer Interaction, Interact'99, Brewster, S., Cawsey, A. & Cockton, G. (eds.), IFIP, 19-20. (1999)
- Gaver, W.W. (1986). Auditory Icons: Using Sound in Computer Interfaces. *Human-Computer Interaction*, **2**, 167-177.
- Gaver, W.W. (1989). The Sonic Finder: An Interface that Uses Auditory Icons. *Human-Computer Interaction*, **4**, 67-94.
- Myers, B., Giuse, D., Dannenberg, R., Zanden, B., Kosbie, D., Pervin, E., Mickish, A. & Marchal, P. (1990). Garnet: Comprehensive Support for Graphical Highly Interactive User Interfaces. *IEEE Computer*, **23**, 71-85.
- Oakley, I., McGee, M., Brewster, S. & Gray, P. *Putting The Feel Into Look And Feel* In: Proceedings of ACM CHI'2000, (eds.), ACM Press, Addison-Wesley, (2000)
- Pfaff, G.E. (1985). *User Interface Management Systems: Proceedings of the Seeheim Workshop*. Berlin: Springer-Verlag.
- Savidis, A. & Stephanidis, C. *Developing Dual Interfaces for Integrating Blind and Sighted Users: The HOMER UIMS* In: Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems, (eds.), ACM Press, Addison-Wesley, 106-113. (1995)
- Scheifler, R.W. & Gettys, J. (1986). The X Window System. *ACM Transactions on Graphics*, **5**, 79-109.

Thevenin, D. & Coutaz, J. *Plasticity of User Interfaces: Framework and Research Agenda* In: Proceedings of Interact'99, Sasse, A. & Johnson, C. (eds.), IFIP, IOS Press, 110-117. (1999)