Voicemail Visualisation from Prototype to Plug-in: Technology Transfer through Component Software

Article · November 2001

Source: CiteSeer

CITATIONS
2

READS
127

4 authors, including:

Steve Loughran
Apache Software Foundation
20 PUBLICATIONS **220** CITATIONS
SEE PROFILE

Roger Tucker
Sonocent
20 PUBLICATIONS **282** CITATIONS
SEE PROFILE

Voicemail Visualisation from Prototype to Plug-in: Technology Transfer through Component Software

Steve Loughran, Nick Haddock and Roger Tucker
Hewlett-Packard Laboratories,
Filton Road
Bristol
BS12 6QZ England
+44 (117) 979 9910
{slo,njh,rcft }@hplb.hpl.hp.com

ABSTRACT
This paper describes how we improved the display of voicemail messages in an email application. Our software was delivered as a drop in "ActiveX Control" for use within another application.

We found that implementing the user interface as a software component was an effective method of transferring our technology from the lab to the field. It enabled the interface to be implemented in two months and integrated without merging source code.

For effective user interaction the whole application had to be seamlessly integrated. We believe that our design consultancy proved as critical as the software.

Keywords
Technology Transfer, User Interface Design, Voicemail,

INTRODUCTION
A problem we have encountered with user interface (UI) innovation is that it is very hard to transfer point improvements in interfaces into systems which would benefit from it. There seem to be a number of reasons for this.

Firstly, the complexity of commercial applications is such that a single programmer or small team of programmers can no longer expect to deliver a complete application within the development schedules of modern products.

Secondly is the trend by many of the groups within our organisation to buy the software from third parties. This systems integration process enables them to rapidly change the mix of the bundled software to support different market segments. It also ensures that the most popular applications can be included in the package.

This trend makes it hard for our research projects to transfer software and interface technology into the products . We can build prototypes to demonstrate how a user's interaction with the system could be improved, and everyone may agree that the benefits are clear and could provide selling points for the product, but unless the ideas can be transferred to the product the investment in research can be viewed as wasted.

Previous transfer routes have often involved re-implementing the new interface in the product code, usually assigning the original designers and programmers to the product implementation. When the software is being written by a third party this method is less effective. Source code becomes a valuable commodity, and legal issues soon complicate any potential transfer route.

We have recently completed the process of transferring an improved UI for viewing received audio messages. The application is provided by an Independent Software Vendor (ISV) and the transfer process was performed without recourse to code sharing or the physical transfer of people. Instead, a new technology was key: ActiveX Controls -one of a number of competing methods of building up applications from component parts. By providing a component which implemented the core of the interface, it was easy for the 3rd party developers to add it to their existing application.

THE PROBLEM
Our research goal was to develop technology which made it easier for users to listen to and extract relevant information from received voice messages. The two common interfaces for processing such messages are currently the answering machine, with its push-button playback, and the telephone-handset interface to voicemail which is rather cumbersome. We were particularly interested in improvements which utilised the dynamic visual display of a PC.

A survey of 3000 heavy voicemail users within Hewlett-Packard [1] provided some insights into how users processed received messages. Most messages were deleted straight after playback -only 5% were archived. Questionnaires sent to eighty of these users revealed that forty percent of messages resulted in notes being taken.

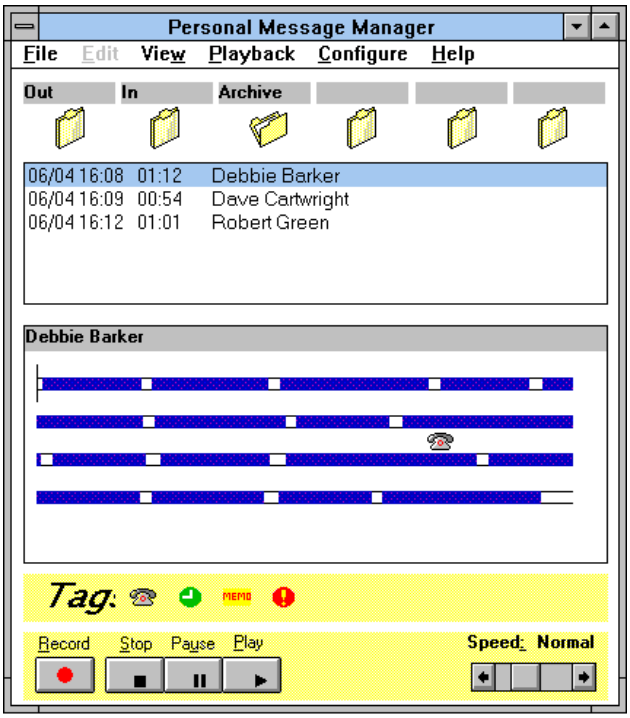

Figure 1: The original prototype

In order to assist note-taking and other information extraction tasks, a key feature which users requested was a reliable textual transcription of the voice message. This would also allow straightforward integration with email, since this was their preferred method of receiving messages (whereas voice was preferred as the medium for generating messages). However, reliable automatic recognition of the kind of spontaneous speech occurring in voice messages is not yet possible. Our project therefore chose to concentrate on simpler techniques for assisting with spoken message review and annotation, and integrating these with an email system.

THE PROTOTYPE

The original prototype "Personal Message Manager" simulated a mailbox for voicemail messages (for a complete description of this research, see [2]). Recorded messages were automatically segmented into blocks roughly corresponding to phrases in the speech. These blocks were then displayed to the user on a graphical timeline, similar to the systems described in [3,4].

The key features of the prototype were:

- Graphical display of duration and structure of a message
- Navigation by clicking on segments within the message
- Highlighting of the current segment during playback
- A "tape recorder" toolbar for simple interaction
- The ability to tag segments with icons and notes to indicate the location of addresses, numbers or important facts.
- The ability to speedup and slowdown message playback while retaining constant pitch.

The system consisted of a Visual Basic application invoking 'C' algorithms in libraries for the audio processing and playback. It effectively demonstrated how a UI for voicemail review and annotation could be designed, without actually doing the engineering work of telephone integration. It was demonstrated internally and used to experiment with various design choices to see which were preferred.

THE PRODUCT

The prototype demonstrated the benefits of assisting audio visualisation. All that remained was to turn it into a product. One of our divisions did actually include in their systems a voice-enabled sound/modem card, and an answering machine application. They were also changing software systems as part of their transition to Windows 95. This operating system included "Exchange Client", an email client intended to provide a single in tray to multiple message transports. The standard installation could support Fax, Internet email and various proprietary email systems. The voicemail add in was to be an extension application which provided a new transport service running in the background and a custom window for displaying received messages. This software was not being developed internally, but was being bought from an Independent Software Vendor.

To enable our project's user interface to enhance the appearance and utility of received messages, we had to transfer it into this new product.

One proposal was to provide a front end to the Exchange client which was more suitable to the display of incoming voice messages. The key problem was that with received email the sender and subject line are often informative, whereas with received voice or facsimile messages, this is not the case. This limits the utility of the "message list" view of the in box. For voice messages, we felt that a display of each message's duration would allow users to see at a glance which messages were short or long, and choose how to deal with them. Methods for playing each message and all messages (answering machine style) would be available. Figure 2 shows a prototype screen design of the voicemail specific window.

The problem with this approach is that we needed to

- Design a complete voice mailbox user interface
- Implement the interface, connecting to the existing Exchange inbox and folders, and integrating with the Windows shell
- Produce documentation and on-line help.

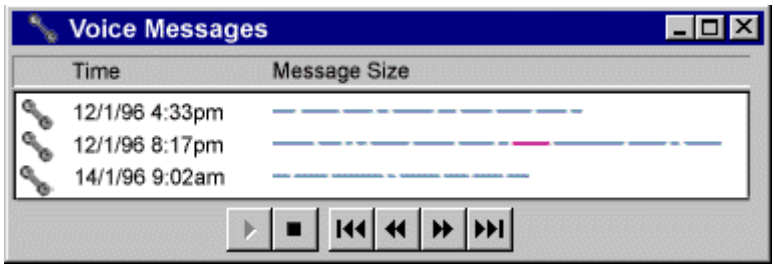

Figure 2: draft design for a voice message box

This, the "traditional" approach to technology transfer, would not have resulted in a usable, working product within the product's schedule. The customer division was also concerned about long term support: any major replacement of the standard system components would incur long term maintenance and telephone support costs. Clearly this method of converting a prototype to an application was not feasible.

We decided that a more effective way of transferring the UI was to re-implement it in an "ActiveX Control", which could then be dropped in to the ISV's existing dialog box design for viewing the messages. We would provide the audio segmentation algorithms and UI for navigation and annotation, and the ISV would provide the audio data and persistent storage of any extra data which we needed to attach to the message. This would leave the standard Exchange application as the top level interface to received messages.

ActiveX Controls

An ActiveX control is a software component which can be re-used by application and web page developers. The control is simply a body of code which communicates with its host application (the "container") via Microsoft's Component Object Model (COM). It must export a number of software interfaces which the container can use to mange its lifestyle, manipulate any exported properties and invoke any available commands. The control must draw itself within the container's window and interact with the user within the confines of its allotted area. It may also send event and error notifications to the container for appropriate processing. As an example, a simple button control, would export a "Caption" property, and provide a "ButtonPressed" event.

A large range of controls are available, commonly to render a specific file format or provide a feature often added to applications -such as graph drawing or text editing. The control architecture has also recently been revised for web pages: an HTML script can contain a reference to a control, is properties and a URL to the origin of the appropriate binaries.

INTERFACE EVOLUTION

Initial Design

The first interface design (Figure 3) was done using Adobe Photoshop. This is a good tool for designing the appearance of an application: designers are not restricted to what is easy to implement. It is ineffective for demonstrating interaction: we resorted to building a storyboard from a sequence of images. The original prototype acted as the demonstrator and test platform for alternative interaction styles.

The control was to convert a supplied audio stream into a segmented timeline, showing speech and silences to the same scale. Appearance options to be left to the container included foreground, background and highlight colours, the heights in pixels of timelines and the gaps between rows, and the milliseconds of audio per pixel.

The task of navigating within the message would be accomplished by clicking in the timeline. A toolbar would offer controls to start, stop and pause playback, move to the next and previous speech chunks, and to add annotations. The toolbar would be hideable, to support alternative designs.

Figure 3: Initial Control Design

Support for a pop-up menu was also to be included, for it offered an effective way of supporting context sensitive operations without taking up valuable display space. Keyboard shortcuts were also to be provided. Rather than implement the pop-up menu and keyboard shortcuts ourselves, we felt that these would be better left to the container. This would allow the container designers to implement a consistent keyboard interface, and provide a localised menu which integrated control commands with container operations. The control therefore had to notify the container whenever a right mouse button click had occurred, and accompany each exported command with a boolean property which was true only when a menu or button invoking that command needed to be enabled.

We had agreed on a staggered delivery schedule of :-

- Audio segmentation, display and playback
- Constant pitch speed adjustment
- Message annotation/tagging

The construction of the original prototype ensured that the issues relating to each feature were already well understood.

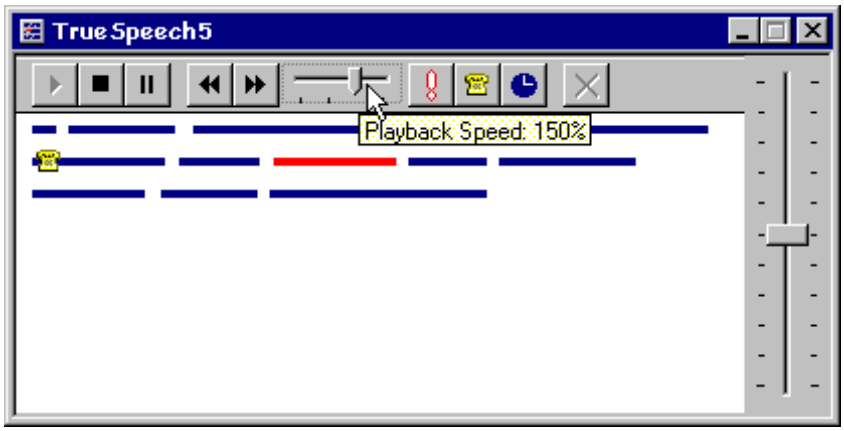

Figure 4: Control in Test Harness

Final Design

Figure 4 shows a view of a test harness application playing back a message. The user has just dragged the horizontal slider to increase the playback rate to 50% faster than normal: the control supports rates from 50% to 200% in 25% increments. The algorithm supported a speedup of 400% but we limited the slider to the most useful portion of the range.

The vertical slider is a volume control provided by the container, which adjusts the volume system wide. Such functionality is available from the system's start bar, but it may make sense to duplicate this functionality in the message window.

The toolbar buttons to the right of the speed slider allow tags to be added and removed from speech chunks. Note that icon design by real artist was performed after these pictures were taken: during development we put together some provisional representations of a telephone, a clock and an exclamation mark -the latter to indicate another point of note. Clicking on any of the buttons adds the appropriate image to that piece of speech. A minor design point was whether to limit the number of tags which a user could add to a single chunk of audio, preventing them from overrunning into the next chunk of speech. We decided not to, even though it complicated the drawing code: we did not want to unduly restrict users' interaction.

The original prototype allowed text notes to be added to message chunks, and allowed searching for messages by the associated text. We dropped this feature from the production code for time and interface complexity reasons. In the absence of this text annotation, it seemed unlikely that users would want to do much marking up of a message, short of noting where an important fact was for immediate replay. Accordingly, the facilities for manipulating the tags was limited to deleting all the tags attached to the current chunk -available from the toolbar- or clearing all tags in a message. The latter is only available if the container provides a mechanism for invoking the command.

The original prototype split audio chunks which extended past the end of a line. The production control implements a chunk wrapping algorithm, so that the message appears more like a piece of ragged right margin text. This looks nicer, although very long chunks still have to be split.

There was one design issue for which we had no right answer: *"whether and how should the display scroll when playing back a long message?"* When a message is displayed which is larger than the control's window, a scrollbar appears, allowing the user to scroll up and down to display the whole message. We were unsure as to whether we should automatically scroll this window during playback. Although the original prototype scrolled the window to keep the playing segment visible, we decided to leave scrolling to the user. Not only was it less confusing, it was slightly easier to implement. We also chose to encourage container implementers to resize the control whenever the container is resized, which allows most voicemail messages to be viewed without resorting to scrolling.

Controls have to be usable during the container's design phase, providing as much as a preview of their run time capabilities as possible. This is to assist the container designers and encourage use of the control. Our control was lacking in this respect -it required an audio message before the timeline was displayed and the toolbar activated. Commercial products targeted at application developers would have to invest more effort in improving the design-time behaviour.

IMPLEMENTATION

The schedule for implementation was pretty tight: the initial delivery deadline was eight weeks from the date when we agreed to build the control. We went from functional specification to software interface specification and first coding within a week.

The initial creation of the control was automated in the compiler via a sequence of dialog boxes: it was important that this sequence was followed after the initial design was completed, as there was no easy way of changing one's mind after the automated code generation sequence. The external programming interface was declared at this time. We were fortunate that very few changes were needed to this interface, as most modifications had the unfortunate side effect of breaking almost every test application. The time spent thinking before coding was clearly vital.

We had worried about how long it would take to migrate the existing audio algorithms from 16 bit code to 32 bit code. In fact this only took an afternoon; more time was spent adding new code to uncompress TrueSpeech encoded audio prior to performing the segmentation. This decompression process took so long that it became a requirement for the voicemail transport to instruct the control to perform this process in the background immediately after a message was received. Once a segmentation index had been derived, the positions stored referred to the original compressed data, so during playback the message segments were decompressed on demand. Supporting background segmentation required a back door to be added to the control so that the voicemail transport service could invoke the process without having to create an instance of the control or make use of COM for inter-process communications.

What turned out to take the most development time was actually the user interface. Not the actual painting of segmented audio messages, or the processing of user mouse clicks, but other aspects, only the first of which was expected to be difficult:-

- Layout of the audio without splitting segments at the of each line.
- Managing a variable height scrollbar which seemingly appeared and disappeared at whim.
- Getting Windows 95 interface components such as toolbar buttons and ToolTips to work

The latter task was inconvenient, as easy access to these features was provided by the C++ class libraries which came with the compiler. Unfortunately most of these classes contained major assumptions about the architecture of the application which effectively rendered them unusable within our control. This could only be rectified by cut-and-paste subclassing of the class library source: an inelegant technique which consumed time and may increase long term maintenance costs.

Internationalisation was critical for a globally available product. Even though the only text was within the ToolTips which pop up above toolbar buttons and a few error messages, we still found that a lot of effort went into internationalisation, specifically into testing: it required localised installations of windows for proper tests. Although UNICODE may simplify parts of the process in future, there are many other cultural variables which can cause problems. Two issues which concerned us were whether messages in different languages needed to be segmented differently and whether users in countries with a right-to-left reading order would prefer audio messages to be displayed similarly.

INTEGRATION

During coding, we were in constant communication via email with the voicemail software vendors, and our customers in the division. The delivery schedule was such that the ISV could not begin integrating our control with their application until after our code was due to be "frozen": after this date only bug fixes could be added, not new features. We resorted to building a pair of test harness applications, to demonstrate the control's features and debug the functionality. Although ActiveX controls can nominally be used in any container, with full language independence, we found it best to use a C++ container as debugging was much easier. We also found it important to verify that the control worked in alternative containers, which did add to the testing process.

Over time the test harness grew to be a demonstration of how to seamlessly integrate control and container, adding keyboard shortcuts and pop-up menus to the basic control. When builds of the control were emailed to the ISV, copies of the test harness -including source- were also shipped out, in order to simplify integration. The build process included automatic delivery of copies to our local web site, where our internal customers could view progress. This proved an effective way for interested parties to track the work's progress without interrupting the engineers.

The control offered various properties which could be manipulated to alter its visual appearance and behaviour This was to enable flexibility of re-use. We felt, however, that this ability to customise the control was dangerous: it could be used to produce an ugly interface. In collaboration with the PC division, we resolved various design issues as best we could, and produced a design guide, describing how to build a good user interface with the component.

When the ISV integrated the control, it went smoothly, with only a few minor enhancements required. To support playback through a telephone handset, rather than attached loudspeakers or headphones, the control needed to provide a "destination" property, and ensure that before playback commenced the handset was off hook but not being used for a telephone call. Rather than add such low level telephony code to the control, the handset manipulation was delegated up to the container. Object interfaces were defined so that a container could provide an interface which would get invoked on the success or failure of any attempt to open a sound output device. When the attempt failed because the handset was on hook, the container would present the user with a dialog box asking them to take the phone off hook. In its present incarnation, this interface is a bit inelegant, as the user must press the single button on the dialog -"OK"- after the handset was lifted. A better implementation would not only automatically dismiss the dialog, it would allow the user to redirect playback to the loudspeaker.

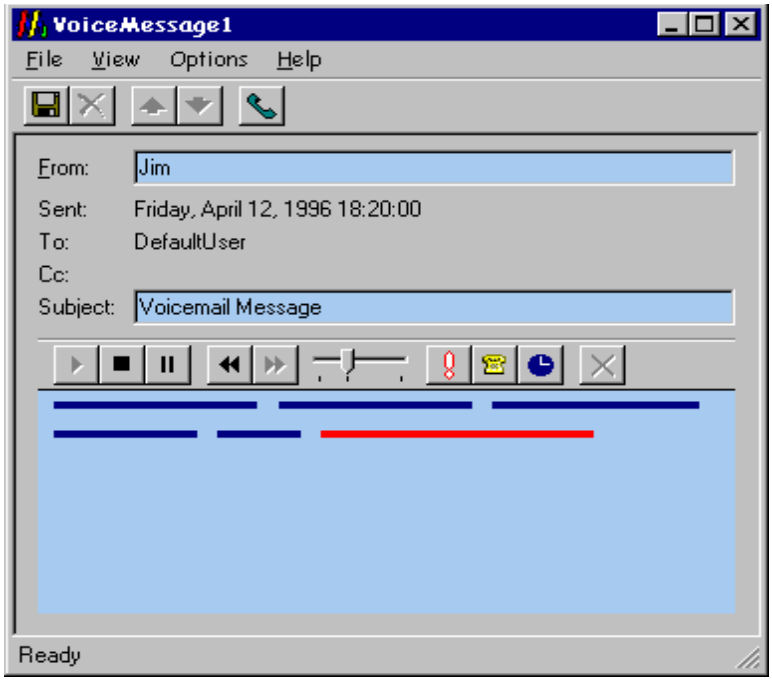

Figure 5: the integrated interface

Our belief is that the provision of design guidelines and an example application is as critical to ensuring that the control can be built into a usable application as providing the control itself. It is worth noting, therefore, that most such controls do not ship with any design tips, merely a programming manual. Perhaps if they did ship with such guidelines then the applications which use them would be improved.

We would have liked to have gained data on the usability of the application by testing it out on naïve users. There was not enough time to do this, and given that we did not construct most of the application it was logistically impossible. Instead we resorted to the subjective opinions of the original prototype designers, our customers in the PC division, and HCI professionals within the Lab. The latter were particularly useful, as they could sit down and play with the application without the corruption of over-familiarity which we had acquired. However, some questions remain. Would users file more voice messages away if the retrieval mechanism were improved? Is the hands-free, one button for all messages playback interface of answering machines actually superior? Only time, and the feedback from the shipping systems, will tell.

FUTURES

The product is available on HP's personal computers as part of a business communications package.

The control has been investigated for use in an experiment in integration with our company-wide voicemail system, so that users can receive all their incoming messages via a single mailbox. We have also demonstrated its creation within web pages, using it to preview audio messages prior to download and playback. However, major re-engineering is really needed to support asynchronous audio download over slow network links.

Some research has also been done into more substantial modifications to the original Personal Message Manager interface. Enhancements were made to the control to facilitate this, but for such a radical change it has actually proved easier to resort to the original prototype. The production control is therefore good when the basic interface can be reused, but there are better ways of prototyping a major overhaul of the interaction.

IMPLEMENTING INTERFACES IN COMPONENTS

There has been a lot of recent publicity about the notion that component software -applets, ActiveX controls, and objects- are going to enable a new generation of applications. Such technology certainly assists in enabling more people to construct applications and interactive web pages. There is always the risk that by empowering a new generation of interface authors, we will end up with a new generation badly designed interfaces. Well designed components accompanied by detailed design guidelines and quality sample applications may help in providing the best of both worlds: quality user interfaces built by people who understand the problem domains.

For our project the component paradigm was clearly critical in ensuring a successful transfer. It did not allow us to develop our control in complete isolation from the application -regular communication and special code enhancements were also critical to the transfer.

We would certainly use this delivery route in the future, provided our designs could be easily packaged up into components which offered drop in user interfaces or simple programmatic interfaces to complex proprietary algorithms. Many Digital Signal Processing tasks fall into the latter category, and many interface enhancements can be placed in the former. However, radical redesigns of entire applications and user interfaces do not seem so suitable to this approach.

We do not intend to prototype new interfaces in ActiveX controls in the near future, because their development process is not well suited to interface prototyping. Not only do many design decisions have to be made in the first day of coding, the extra complexity of testing and debugging the controls unduly increases development time. Hopefully more flexible development environments will emerge which allow component software systems to

be as useful when prototyping interfaces as transferring them.

CONCLUSIONS

- Implementing UI designs as re-usable software components enables them to be put to many uses, far easier than with previous technology transfer methods
- Because such components are simpler than whole applications, they can be built more rapidly and used to enhance existing products.
- The software isn't enough, an interface style guide and design support is critical.

ACKNOWLEDGMENTS

Francois Xavier Lecarpentier, Yan Leshinsky, and Mike Collins are all appreciated for their role in the successful prototyping, implementation and transfer of the product.

REFERENCES

[1] Geelhoed, E., Duhoo, C. VoiceMail: Deal with it here and now! 1995. Available from eg@hplb.hpl.hp.com

[2] Haddock, N.J. and Tucker, R.C.F. The Personal Message Manager: Integrated Techniques for Reviewing, Indexing and Structuring Voice Recordings. Manuscript in preparation.

[3] Hindus, D.C. Schmandt & C.Horner. 1993. Capturing, Structuring, and Representing Ubiquitous Audio. ACM Transactions on Information Systems 11(4), 376-400.

[4] Ades, S. & D.C. Swinehart. 1986.Voice Annotation and Editing in a Workstation Environment. In the proceedings of the 1986 Conference: The American Voice I/O Society, San Jose California, 13-28.