CONSISTENT WINDOWING INTERFACES IN DISTRIBUTED HETEROGENEOUS ENVIRONMENTS

Daniel Owen and Alasdair Rawsthorne

Department of Computer Science,
The University of Manchester,
Oxford Road, Manchester, UK,
M13 9PL.

{owend, alasdair}@cs.man.ac.uk

Abstract: The majority of windowing systems provide a single, unique "look and feel" which applications must assume. Users of multi-platform environments are forced to accept that applications appear and behave differently depending on the platform. This paper describes DHE-Win, an open distributed software architecture that allows users, even in heterogeneous environments, to control the appearance and behaviour of applications. DHE-Win allows users of multi-platform environments to benefit from applications having a consistent appearance and behaviour.

Furthermore, it is shown that DHE-Win architecture is radically simpler to implement than existing windowing toolkits, thanks to its highly modular structure and separation of policy and mechanism.

Keywords: Customisable, distributed, heterogeneous, multi-platform, user interface, windowing system

1 INTRODUCTION

The majority of windowing systems provide a single, unique "look and feel" which applications must assume. X Windows is more flexible in that a limited number of toolkits are provided together with selective customisable features. Nevertheless most users are dictated the interface they must use, irrespective of whether they like it. This situation is worsened for users of multi-platform environments, as they are

L. Kutvonen et al. (eds.), *Distributed Applications and Interoperable Systems II*

forced to accept that applications appear and behave differently depending on the platform.

This paper describes DHE-Win, an open distributed software architecture that allows users, even in heterogeneous environments, to control the appearance and behaviour of applications. Users define the "look and feel" they want by creating description modules. The modules are provided in a platform-neutral manner and can be published for use over a distributed environment allowing access by any machine. This allows users of multi-platform environments to benefit from applications having a consistent appearance and behaviour. DHE-win can also be used to provide "look and feel" descriptions tailored for a particular user or environment. For example specialised widgets could be designed for use with the partially sighted.

Current windowing software is large and complex with highly integrated functionality. These features conflict with the requirements of a configurable windowing system. It was necessary to develop a methodology that separated out functionality and in doing so simplified the structure of the software, allowing users to define their own personal "look and feel" without considering the remainder of the system.

The desired approach was achieved by identifying and separating the windowing system policy of how the functionality is achieved, and the mechanism through which it is provided. By examining existing windowing systems it can be seen that at a semantic level, all consist of widgets (a generic term for a window, button, etc.) and a set of functions such as move, resize, and draw line, which are applied to these widgets. Systems differ only in the way these widgets and the set of related functions are used. Therefore in terms of mechanism and policy, the mechanism refers to the widgets and set of functions that can be applied to them, whilst the policy relates to how these functions are used to achieve the required appearance and behaviour. This approach has enabled us to create an open framework, which can provide consistent windowing system functionality across heterogeneous environments.

DHE-Win was tested by constructing an example implementation designed to run on top of the X Window System [1]. The implementation used a CORBA-related software tool, known as Inter-Language Unification [2], and is currently capable of executing programs written for three different windowing toolkits, XView, Athena, and Lesstif (a Motif implementation). Using DHE-Win programs written and compiled for these toolkits can have their "look and feel" changed to assume any appearance and behaviour, including that of other toolkits.

2 POLICY AND MECHANISM IN A WINDOWING SYSTEM

The separation of mechanism and policy in operating system design has been widely reported [3]. The mechanism usually represents an abstraction of the underlying hardware together with a set of basic functions for manipulating this hardware abstraction. Policy refers to how this functionality is used to generate a variety of different behaviours. For example, activities such as reading/writing a block of memory or adding/removing a process from the CPU are considered as mechanism, while facilities that make use of the mechanism, such as file caching decisions or different process scheduling algorithms are considered as policy.

Applying these techniques to a window system, this work uses mechanism as a single type of object, known as a base-widget (in many windowing systems it is

referred to as a window). The base-widget is used simply to represent an area of the computer's display, onto which lines etc. can be drawn. It is important not to confuse the base-widget with that of high-level windows, which contains borders and other functionality, such as resize buttons etc. In DHE-Win, these high-level windows are built using a number of the base-widgets.

The base-widget object is comparable to that of the abstract hardware found in customisable operating systems, and in a similar manner has a basic set of functions that can be applied to it. These functions can be divided into two categories, those that provide painting facilities to the widget, and those that control the widget. The painting functions provide the mechanisms needed to draw and produce text etc., while the control functions provide the mechanisms necessary to manipulate base-widgets and include such functionality as create and move.

The window system policy defines how the mechanisms, provided by the base-widget and its functions, are used. The policy decides such things as what base-widgets are created and what appearance and behaviour they should have, for example if defines a widget's shape, size and layout etc.

3 THE DHE-WIN FRAMEWORK

The aim of the research was to provide an open framework that could be used to construct software that allows users to control the appearance and behaviour of applications. The DHE-Win framework is designed to be applicable across a heterogeneous network allowing applications to assume the same "look and feel" regardless of a machine's native windowing system.

3.1 The Architecture

As shown in figure 1, DHE-Win can be divided into three main sections: Application Front-End, Mechanism and Policies.

The Application Front-End consists of two components, the configure module and the call transformation software. The call transformation software intercepts an

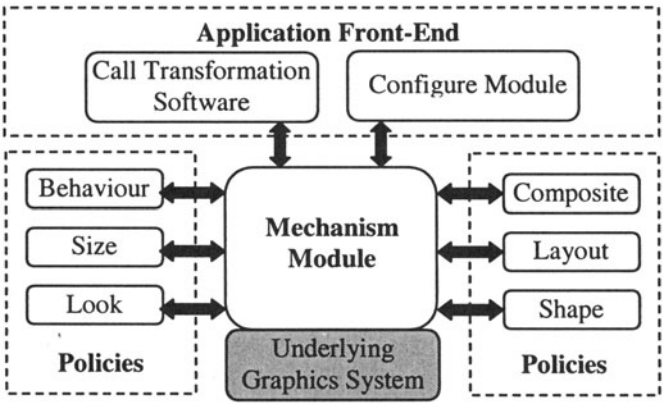

Figure 1. Architecture of the Framework.

application's windowing function calls and transforms them into a neutral format. These calls are then passed onto the mechanism section of the architecture where they are processed.

To give users full control over the appearance and behaviour of applications they must be able to select different policies for different instances of an application. In other words if two copies of the same application binary were executed each could assume its own "look and feel". The greatest benefit that this provides is on a multi-user system, where each user can choose the "look and feel" they require for a particular application. This freedom in choice is provided through the use of the configure module.

The mechanism module provides the core of the framework. Its main task is to provide an interface to a set of tools through which the "look and feel" descriptions can be implemented. The tools are fundamental to any graphical environment and so can be guaranteed to be available on any platform.

The mechanism module also retrieves all events generated by a platform's underlying graphical environment and converts them into a neutral format so the behaviour policy module can act on them. These events range from notifying that the pointer has moved, to expose events that are generated when a widget containing graphical data, such as lines and text, becomes visible.

In addition to providing a platform-neutral interface the mechanism module is responsible for managing all of the widgets instantiated during the running of an application. This includes maintaining widget identifiers and widget attributes, such as current size etc.

The final component of the DHE-Win architecture provides the "look and feel" descriptions. These are presented in the form of policy modules. Each module describes a particular section of functionality used in the creation of an application's appearance and behaviour.

DHE-Win is based around a distributed architecture, allowing various configurations to be implemented. At one extreme all the components described could be duplicated for every machine in the network. However a more desirable approach is to take advantage of the facilities that permit the sharing of components. The distributed approach presents a number of advantages, including:

1. Component-Reuse. It is unlikely that each user would consider developing their personal "look and feel" from scratch. Through the use of a repository various appearances and behaviours can be stored from which users can select and alter as required, to generate the "look and feel" they want.
2. "Look and feel" descriptions can be reached from any machine in the network. This means users have only to select a "look and feel" once, and can use it to control the appearance and behaviour of applications on any platform.
3. Users can experiment with a variety of "look and feels" by simply binding to different policy modules. Through the use of dynamic binding, applications can even have their appearance and behaviour altered at run-time.

3.2 Policy Modules

Policy modules are required to describe how the basic tools provided by the mechanism module are used to construct windowing toolkits. Their primary role is to

define how applications should appear and behave, ranging from the shape and size of widgets, to the actions invoked when a pointer-button is pressed. Each module is responsible for providing a particular type of functionality, and by using different policies it is possible to generate any "look and feel".

One of the most interesting policy modules controls the behaviour of widgets. It controls for example, what should happen if the left-button on the pointer is pressed, or if an expose event is received.

The events are converted into a system-neutral form by the mechanism module and delivered on an individual basis to the behaviour policy, ready for processing. Depending on what event was received and how the policy module is defined, this processing can range from discarding the event to initiating a large chain of response.

The policy module can use any of the data fields provided by the event given, and like the other modules can use functions provided by other policy modules, the mechanism module and even the Application Front-End software. This permits a great freedom in the kind of behaviour that can be provided.

When an event is received by the behaviour policy the relevant information is extracted and compared with that held in the policy module. Once a match has been found the respective policy decisions can be identified allowing the correct behaviour to be provided.

Policy Module	Description of Policy Modules
Composite	Describes how high-level application widgets are constructed using base-widgets, refer to section 3.4.
Size	Defines the size of widgets. The policy can include such information as a widget's default, maximum and minimum size.
Shape	Defines the shape of widgets.
Look	Defines the colour of widgets. The policy is also responsible for describing any bitmaps, text or lines etc. that are to be drawn on the widgets.
Layout	This policy describes the relative position of each base-widget used in the construction of an application widget. In addition it is also a requirement of the layout policy to provide application level layout descriptions. These types of descriptions are necessary when application widgets are themselves required to contain other application widgets. For example, if an application window is to contain buttons, it states where the buttons are to be placed.

Table 1. Summary of Policy Modules.

In addition to the behaviour module the framework defines Composite, Size, Shape, Look and Layout policy modules. A summary of the functionality provided by each is given in Table 1.

3.3 Translating Existing Applications

Software developers can use DHE-Win to provide a "look and feel" for the applications they are developing. Furthermore as DHE-Win is extensible, any type of widget can be invented and incorporated into the application being produced. Traditionally developers would have found it difficult to implement widgets not provided by the platform's windowing toolkit.

However, DHE-Win would be of limited use if restricted to applications that have been written and compiled specifically for the system. A technique has therefore been devised by which the appearance and behaviour of existing applications can be altered.

Applications communicate with underlying system software, including the windowing system through the use of Application Program Interface, (API) calls. Therefore if applications written and compiled for proprietary windowing systems are to have their "look and feel" altered all of the relevant calls have to be intercepted and converted into a form understood by the policy modules.

Call interception is achieved by replacing the native windowing toolkit library with call transformation software provided by DHE-Win. Replacing a library requires techniques specific to a particular platform. However one possible method is to create a replacement library with the same name and have the underlying system's dynamic loader refer to it instead of the original.

3.4 Constructing Application Widgets

In traditional windowing systems the set of application widgets that can be used is predefined. This can be clearly demonstrated in those systems that contain API calls naming specific widgets, such as CreateButton or CreateScrollbar. The DHE-Win framework is quite different. It does not define any application widgets and does not enforce any restrictions on the type of widgets that can be created. The decision on what widgets are provided is left entirely up to the users of the framework who write the necessary policy modules to achieve the "look and feel" they require.

As DHE-Win only provides the base-widget, all high-level application widgets are constructed, as described in the composite policy module, by using this single type. This technique has the advantage that it enables any widget, no matter how distinctive, to be created. If high-level widgets providing facilities such as buttons and scrollbars had been defined, to support applications written for potentially any platform all possible types of widget would have had to be included. Furthermore it is likely that certain types of widget would vary between platforms. For example one system's scrollbar may consist of a single slider button, while others may contain additional buttons. If DHE-Win defined high-level widgets all these variations would have to be included.

3.5 Using Application Widgets

To make use of application widgets it is necessary to distinguish between the different types. For example, the call transformation software needs to be able to state what type of widget should be created, i.e. whether it should be a button or scrollbar etc.

Naming application widgets presents another problem: policy modules need to identify the base-widgets that go into making an application widget. This is required so that the appearance and behaviour of base-widgets can be controlled on an individual basis. In the case of the scrollbar example, the "top button" moves the thumb down when pressed, while the "bottom button" moves the thumb up. If separate parts of an application widget could not be identified this type of behaviour could not be provided.

To satisfy both of these requirements two types of classifier have been defined, referred to as the class and sub-class. The class is used to define the type of application widget, while the sub-class refers to the base-widgets used in its construction. Referring again to the scrollbar example, which consisted of four base-widgets, the sub-classes are named as Back, Top Button, Bottom Button, and Thumb. The class usually describes the widget's functionality, which in this case is Scrollbar. However, it is important to note that classes do not have to be the same or even similar to the names used in original windowing systems. This decision is left entirely up to the implementor of the widget.

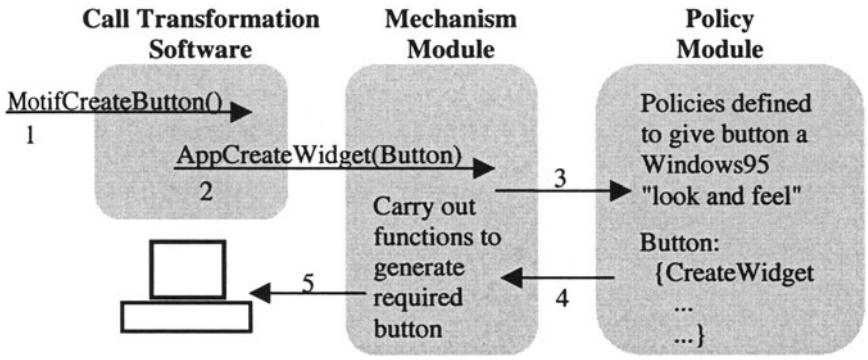

Figure 2. Button Creation Operations.

The way application widgets are used in DHE-Win is best demonstrated through a simple example. This example involves a Motif application appearing as if it had been written under Windows95, and concentrates on providing the functionality of a button. Firstly the call transformation software, which resided in the Application Front-End, traps the Motif call that creates a button. This is then translated into a call to the mechanism module. The call states that an application widget of class "Button" is to be created. The module will then call the policy modules, which contain information relating to how the application class should appear, which in this case is in the form of a Windows95 button. These modules make the necessary calls to the mechanism module so as to create the required widget. This sequence of operations is represented pictorially in figure 2. It is important to note that in order to generate a different "look and feel", different classes are not stated in the call

transformation software, because this would mean altering the software every time a different appearance and behaviour was required. The variation in "look and feel" is instead achieved by replacing policy modules.

4 AN IMPLEMENTATION

Validation of the DHE-Win concepts was achieved by constructing a prototype implementation. The X Window System [1], provided the underlying graphical environment used by the mechanism module. The distributed platform was provided by a CORBA-based architecture, known as Inter-Language Unification [2].

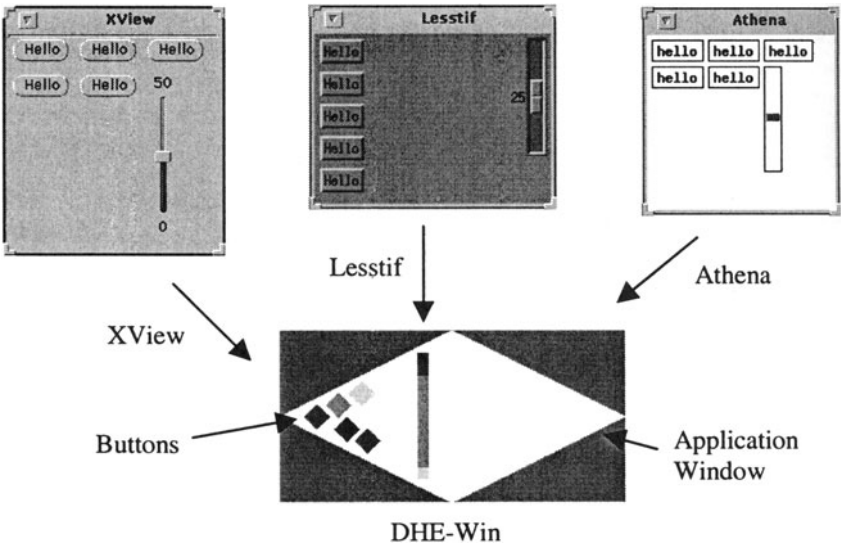

Figure 3. Screen-Shots of the Native Windowing Systems and the Mapping to Generate a Consistent "Look and Feel".

Support for three different windowing toolkits was implemented to demonstrate how DHE-Win could provide a consistent "look and feel" over programs that use different Application Program Interfaces. Figure 3 presents screen-shots of an example application using the three native windowing toolkits and their appearance under DHE-Win. The highly distinctive "look and feel" was chosen to highlight the flexibility in the appearance applications could be given.

Using the example framework implementation it was demonstrated that:

- Applications could assume any "look and feel".
- A user's "look and feel" could be applied in a consistent manner to applications that have been written and compiled for different windowing toolkits.
- Application binaries could have their appearance and behaviour altered.

- DHE-Win components could reside in their own separate address space, allowing access over a network.
- DHE-Win components could be shared by multiple applications native to a variety of windowing toolkits.

Due to time constraints it was not possible to validate all of the facilities demanded by an implementation of the framework. Most noticeably the configure module was missing. Nevertheless experience gained during the implementation suggests that this and other missing functionality, could be provided at a reasonable cost.

5 EVALUATION

For the DHE-Win approach to the implementation of distributed heterogeneous windowing systems to be practical, the performance must be within acceptable limits. In addition, if developers are to provide their own "look and feel" descriptions, it is important that code is kept to a minimum and provided in a structured manner.

5.1 Cost of Framework

To evaluate the performance overhead of DHE-Win, two types of experiment were performed. The first measured the time taken to create a varying number of widgets, while the second measured the response-time latency that applications provided users.

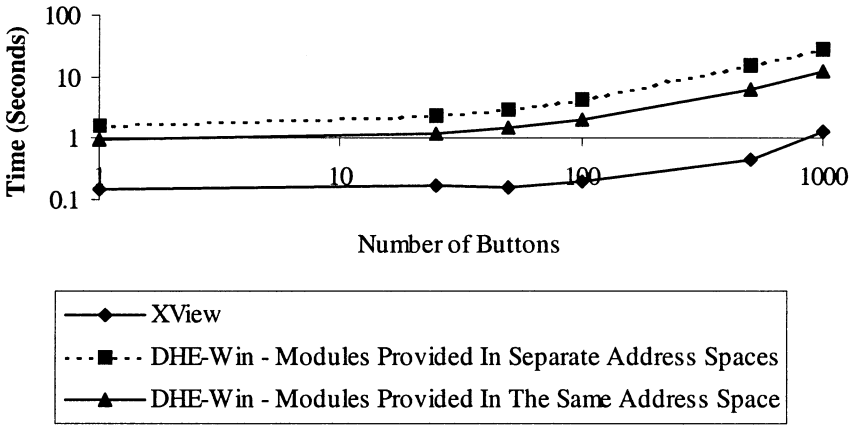

Figure 4. Button Creation Results.

Figure 4 presents results for the widget creation experiment when implemented on a Linux platform, (a SunOS implementation revealed a similar pattern in performance). The experiment compared the native XView toolkit with two forms of the DHE-Win framework. For the "separate" form each of the policy modules were provided in their own separate address space, permitting possible shared use over a network. The "same" form provided all the system in one single address space.

After profiling Win-DHE it was identified that approximately 30% of the total time was spent manipulating widget identifier translation tables. We expect that performance will be improved with the introduction of hash tables.

The implementation of widget shape descriptions was also found to be costly. Currently shapes are described in terms of bitmaps. Although allowing freedom in the description of shapes the operations involved are expensive. By using an alternative technique, for example describing shapes in terms of basic primitives, we expect to achieve further improvements in performance.

Time is also spent performing ILU functionality. This increases as more modules are provided in their own separate address space. By retrieving and caching the required policy modules locally, it would be possible to execute them all in the same address space, thereby eliminating the costs of inter process and network communication.

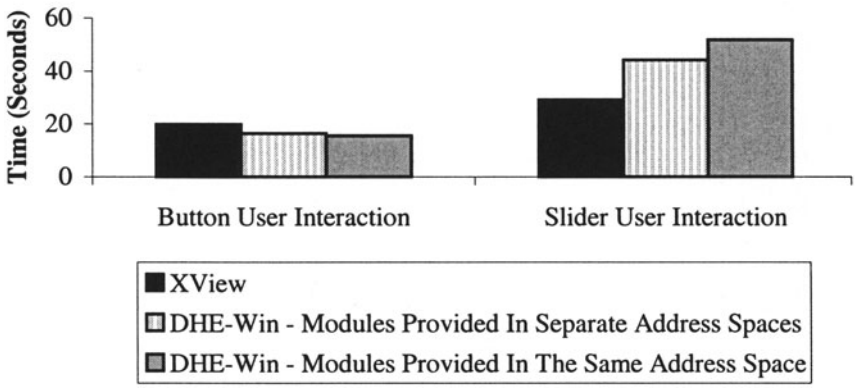

Figure 5. User Interaction Results.

To evaluate the response of user interaction two experiments were performed. The first measured the time taken for a user to press a button 100 times. The second measured the time taken to move a slider's thumb from the minimum to the maximum position and back again 100 times. Figure 5 presents the results for both experiments when performed on the Linux platform.

For the button experiment DHE-Win's performance is comparable to that of the native XView. The extra time taken to perform the slider experiment is believed to be due to accepting too many events generated by the mouse. Under SunOS the response for both of the user interaction experiments was slower, (approximately two times slower for the "same" form and three times slower for the "separate" form). An increase in the number of instructions executed may provide an explanation for the slowdown. The machine running Linux, (Pentium 90), is able to cope with the additional instructions within the time it takes the user to press the button, or move the slider. However for the SunOS machine, (an old SPARC IPC), this is not true.

5.2 Complexity of Software

The measure of software complexity was achieved by summing the number of lines of source code required to provide a particular widget. Table 2 presents the amount of code specifically used to add a button and slider widget to the various windowing toolkits.

Widget	Lesstif	Athena	Xview	DHE-Win
Button	1417	850	755	127
Slider	1796	1026	1956	593

Table 2. Lines of Code Required to Specifically add a Button and Slider Widget.

The results show DHE-Win requires substantially less code than the native windowing toolkits. It is believed that the reasons for this can be explained by the underlying structure of the toolkits. In the case of XView the software was based around a hierarchy of objects. This hierarchy made it difficult to separate out the functionality required by the various widgets. The task was not aided by the large size of the toolkit, (approximately 200,000 lines).

Athena and Lesstif utilise functionality provided by the Xt Instrinstics toolkit [4]. Xt is used to provide the functionality on which the creation of the widget toolkit can be based. Athena and Lesstif define what widgets are created and how they will "look and feel".

This appears to be similar in design to the framework, however many important differences exist. It may be that DHE-Win widgets require much less code then Xt-based toolkits since:

1. Functionality provided by Xt forces a prescriptive structure on how Athena and Lesstif create widgets. DHE-Win, on the other hand, simply provides a set of tools that can be used at the developers' discretion.
2. Xt allows the application user to change certain attributes of a widget, which can relate to the widgets "look and feel", through the use of resources. However, their use introduces two major drawbacks. Firstly, the resources available to the application user are limited. Secondly, incorporating a resource into a widget can greatly increase and complicate the source code, because all possible options have to be implemented, even though it is likely that only one will ever be used. Under DHE-Win the change in "look and feel" would be accomplished by

replacing the policy. This enables any degree of changes to be made, and also keeps the amount of code for a particular widget implementation to a minimum.

6 RELATED WORK

The most related work to the DHE-Win framework is that of single-source development toolkits. These toolkits are aimed at software developers who wish their software to be available on many different platforms. The toolkits allow code written for a particular software platform to be compiled for use on various alternatives, without any (or minimal) changes to the source code. The majority of systems also permit applications to assume one of a limited number of standard "look and feels".

Wind/U [5], developed by Bristol Technology Inc., is a single-source development toolkit. It allows Microsoft Windows applications to be compiled for UNIX and OpenVMS environments. The applications ported can assume the "look and feel" of either Microsoft Windows or Motif, and are able to make use of functionality unique to Windows through the provision of additional emulation libraries.

A set of cross-platform tools, known collectively as the MainWin Studio [6], provides similar functionality. A Microsoft Windows interface is achieved under UNIX by using X windows for top-level window management. All of the child window management is provided through the use of native Windows NT code, which has been compiled for use with UNIX. MainWin permits users to select, at run-time, either a Windows or Motif "look and feel".

The Willows toolkit [7], developed by Willows Software Inc. has a similar architecture to the presented framework in that it provides a platform-neutral layer. However as in the two previous examples it only supports the development of Microsoft Windows' applications. Furthermore applications can only assume one of a limited number of appearances and behaviours. The DHE-Win framework, on the other hand, is designed to support application binaries native to any windowing system, and through the use of policy modules, applications can assume any "look and feel".

A message-based User Interface Management System (UIMS), known as OpenUI [8], developed by Open Software Associate, provides a possible solution to providing user interfaces across heterogeneous environment. However, unlike DHE-Win, applications must assume the "look and feel" native to the platform where the UMIS is currently residing. For example, if the UIMS is being executed under Windows95, the application will assume a Windows95 "look and feel".

This restriction arises because "look and feel" descriptions are given in terms of high-level windowing functions native to the host platform. DHE-Win policy descriptions are platform neutral, which permits any "look and feel" to be defined and used on any platform.

7 CONCLUSION

By separating a windowing system's functionality into policy and mechanism, and by using a distributed modular structure, it has been possible to construct a framework that provides applications with a consistent "look and feel" in heterogeneous environments.

The premise on which the research was based is that advances in modern operating system design can be applied with equal success to this problem. It is therefore interesting to observe how the evolution of windowing systems is compared with that of operating system design, and to note the similarity in structure.

Traditional windowing systems are large complex pieces of software, with no strong internal structure. This type of architecture is comparable to that of the traditional monolithic operating systems. Evolution in operating system design led towards the development of a micro-kernel, where a modular, client-server architecture is used. Once again this can be mapped within the area of windowing systems, with the introduction of such systems, as X Windows.

The latest development in operating system design is that of the nano-kernel. This type of system is characterised by making use of the separation of policy and mechanism, together with a highly modular structure. Since the DHE-Win framework uses these features, it is suggested that it can provide the final evolution in windowing system design.

7.1 Validation of the Framework

It was shown that the DHE-Win architecture promoted self-contained modules, allowing easy identification of the functionality provided. This allowed the software to be extended or altered without disrupting other parts of the system. It was also shown that DHE-Win was radically simpler in design than existing windowing toolkits.

The performance results indicate some overhead in providing a distributed, modular architecture. Nevertheless from the experiments carried out it was shown that the DHE-Win framework could be implemented at an acceptable cost.

To conclude, this work has demonstrated that it is possible to give users control over the appearance and behaviour of applications. Furthermore by providing open distributed architecture the chosen "look and feel" can be applied to applications on any platform.

It is hoped that this research will encourage a future where "look and feel" descriptions can be purchased in much the same way as applications are today. Various policy modules could be bought, allowing applications to assume a variety of appearances and behaviours; the choice is left entirely to the application user.

7.2 Further Work

Although it has been demonstrated that DHE-Win's policy modules require substantially less code to implement than traditional windowing toolkits it is unlikely that users could, or would want to implement them from scratch. Techniques to aid in providing personalised policy modules, such as the provision of core modules, the customisation of existing modules and use of additional software tools requires further investigation.

A subject closely related to windowing support is that of compound document architectures, such as Microsoft's OLE [9], and OpenDoc [10]. These have yet to be addressed in DHE-Win. In addition, with the introduction of multimedia, many APIs

also provide support for video, sound etc. If the framework is to be used in a commercial environment, these additional facilities must be incorporated.

8 ACKNOWLEDGEMENTS

The authors would like to acknowledge the referees for their helpful comments, which improve the paper.

References

[1] QUERCIA V., O'REILLY T., *X Window System User's Guide Second Edition*, O'Reilly and Associates Inc., October 1989.

[2] JANSSEN B., SPREITZER M. et al, *ILU 2.0alpha8 Reference Manual*, Xerox Corporation, July 1996.

[3] MAYES, K., *Trends in Operating Systems Towards Dynamic User-level Policy Provision*, Department of Computer Science Technical Report UMCS-93-9-1, University of Manchester, September 1993.

[4] NYE A., O'REILLY T., *X Toolkit Intrinsics Programming Manual Motif Edition*, O'Reilly and Associates Inc., February 1995.

[5] BRISTOL TECHNOLOGY INC. Wind/U White Paper, Bristol Technology Inc., 241 Ethan Allen Highway, Ridgefield, Connecticut, USA, 1996. URL http://www.bristol.com.

[6] MAINSOFT MainWin Studio, Mainsoft, 1270 Oakmead Parkway, Suite 310, California, USA. URL http://www.mainsoft.com.

[7] WILLOWS SOFTWARE INC. The Willows Toolkit Technical White Paper, Willows Software Inc., 1996. URL http://www.willows.com.

[8] OPEN SOFTWARE ASSOCIATES OpenUI, Open Software Associates, Nashua, New Hampshire, USA, 1996. URL http://www.osa.com.

[9] WAYNER, P. Objects on the March, BYTE, Vol. 19, no. 1, pp. 139-150, January 1994.

[10] NELSON, C. OpenDoc and Its Architecture, Proceedings of the 9th Annual X Technical Conference, pp. 107-126, Boston, Massachusetts, USA, January 30- February 1, 1995.

Biographies

Alasdair Rawsthorne is a Lecturer in Computer Science at The University of Manchester, England. His research interests include: compilers, operating systems, and computer architecture.

Daniel Owen is studying for a PhD in Computer Science at The University of Manchester.