A SERVO APPLICATION DEVELOPMENT ENVIRONMENT FOR MICROSOFT WINDOWS ®

J. Randolph Andrews
Douloi Automation
740 Camden Avenue Suite A1
Campbell, CA 95008-4102
(408) 374-6322

Paper presented at the
1992 Incremental Motion
Control Systems and
Devices Symposium

Introduction

Microsoft Windows® has emerged as a standard for user interfaces on the IBM PC. Although simple and familiar to the user of an application, Windows can be intimidating for motion application developers. The inherent complexity of Windows is additionally complicated by real time issues necessary for solving motion control problems. Windows does not, by itself, work well for real time applications.

An application development environment named "Servo Application Workbench" (SAW) is described which simplifies the creation of high performance motion control Windows applications.

The Servo Application Workbench augments Windows with a multitasker that addresses the real time limitations of Windows. A motion system is included that can coordinate 1 to 6 servo axis through continuous paths and perform motion profile "splicing". A "Screen Painter" is included that allows the user to create a control panel for the application. Using a "clip art" metaphor the user can browse through software "catalogs" of pre-fabricated subassemblies and paste selections into the application. Additional program behaviors not associated with the catalog parts can be added to the application with the built in Object Based Pascal language.

The language system allows the user to run multiple programs at the same time simplifying the construction of advanced applications. Multitasking programs can communicate to the control panel that has been painted, the motion system, IO boards in the PC, and each other. The language allows the user to describe application related objects and to combine these object descriptions with related operations to manipulate them. The language includes a "try..recover" style structured exception handling mechanism that allows simpler management of corrective procedures when errors are detected.

The Servo Application Workbench permits developers to quickly and simply create high quality and performance real time Windows motion control applications. An example telemanipulator application will be described to illustrate how an application environment can simplify development.

The following sections include a description of an example application, application construction, discussion of the development environment contribution, and summary.

Example Application

To provide a context for discussing motion application development consider the following telemanipulator application.

A telemanipulator system is composed of two kinematically equivalent backdrivable mechanisms. A motion application is needed to provide the following behaviors:

Position Tracking Mode

In position tracking mode the slave manipulator follows the positions of the master. The master mechanism is passive being used as a multi-dimensional joystick to direct the servoing slave mechanism.

Force Reflection Mode

In force reflection mode both master and slave mechanisms are under servo control. Pushing against the master mechanism causes the master to move and the slave to follow. Pushing against the slave mechanism causes the slave to move and the master to follow. The system behaves as if mechanical shafts were connecting master mechanism joints to the corresponding slave mechanism joints. This bi-directional causality implies that forces encountered by the slave are felt by the operator providing the operator with reflected force information. This is a type of multiple-input, multiple-output problem since the directives to both the master and slave servos are related to the state of both the master and slave mechanisms.

Position Displays

The control panel should provide continuous display of the location of both the master and slave mechanisms.

Error Display Graph

A graphical display of the position difference between the master and slave would be useful for confirming good tracking behavior.

Abort Button

A means of ending the current mode and disabling the servos is required in order to quickly shutdown the system.

Figure 1 shows a motion application which displays these behaviors. The construction of this application will be described in the following sections. The application behaves like other Microsoft Windows applications such as dragging by the title bar and displaying system options when the upper left icon is selected. The current telemanipulator mode and mechanism positions are seen on the main panel. The error display is shown "popped up" on top of the main Telemanipulator Control window.

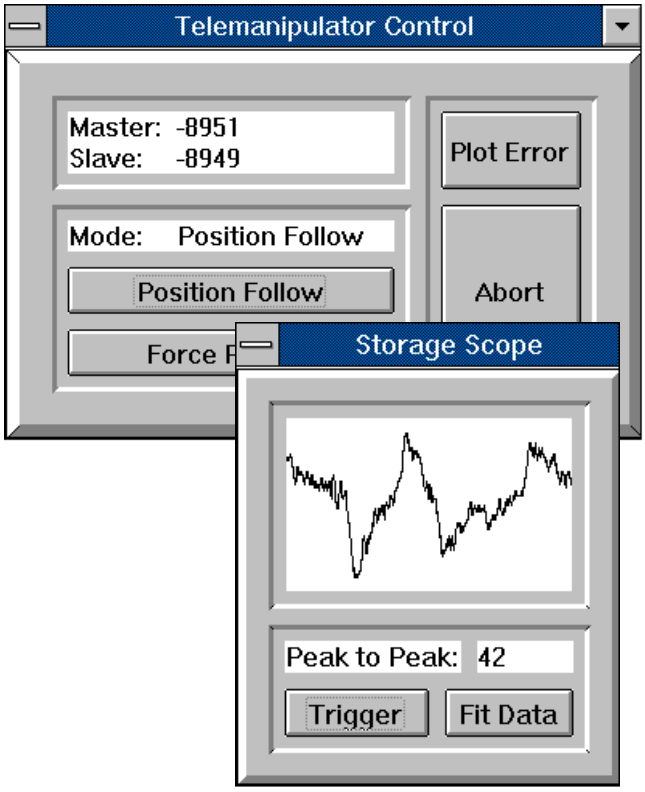

Figure 1. Telemanipulator Application

Force Reflection Approach

This example application uses a force reflection approach based on the elastic shaft model shown in figure 2.

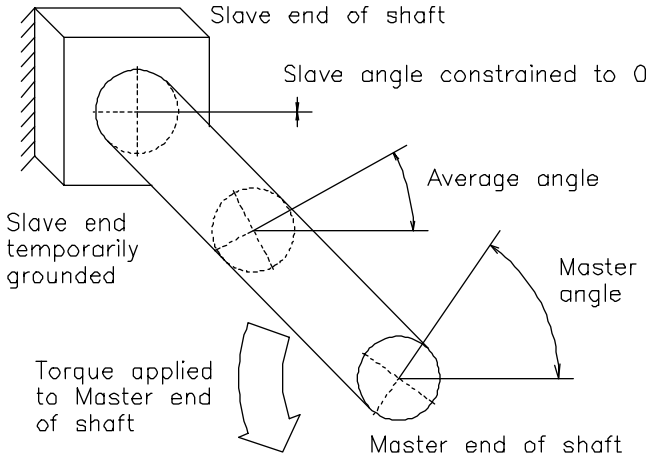

Figure 2. Elastic Shaft Model

One end of the shaft represents a master joint and the other end of the shaft represents the corresponding slave joint. Consider the case that the slave has encountered an obstacle and, constrained not to rotate, is temporarily grounded. A torque applied to the master end of the shaft causes an angular strain, theta, proportional to that applied torque. Half way down the length of the shaft a strain of Theta/2 has occurred. The "point of symmetry" in this problem is the middle of the shaft since either mechanism can direct the system behavior in force reflection mode. By establishing a reference frame along the rotated theta/2 section of the shaft midpoint it can be seen that each end of the shaft is attempting to elastically return to 0 angular displacement with respect to this frame.

The most fundamental behavior of a position servo is to apply a restoring torque proportional to an angular displacement from a commanded set point. Accordingly, one simple implementation of bi-directional force reflection is to use a PD position servo for each mechanism and continually command each servo to move to the average position of the two.

Application Construction

The following sections describe the principles and metaphors used in constructing an application with the Servo Application Workbench.

SAW Overview

Figure 3 shows the appearance of the Servo Application Workbench ready to build an application. The SAW main window contains a drawing area with two rows of "tools" on the left side. The left tool row creates graphic images to place in the application such as colored rectangles, circles, and raised surfaces. The right tool row creates Windows controls such as buttons and editors as well as specialized controls such as plotters. A default plate is seen in the drawing area.

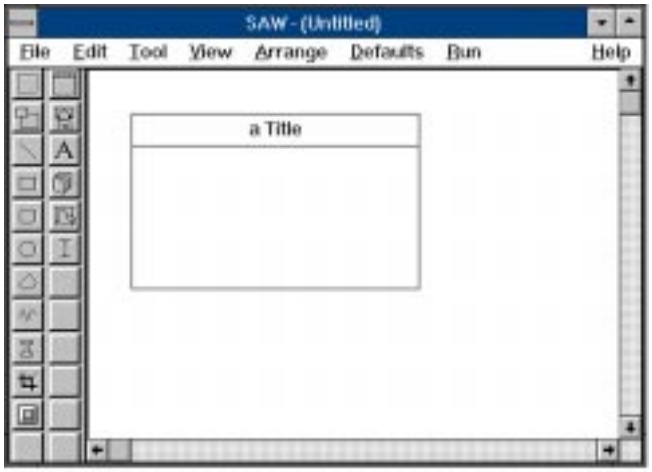

Figure 3. SAW Overview

The Plate Metaphor

SAW uses several different metaphors to convey concepts to the developer. One metaphor is the idea of a "plate". When building a conventional control panel, holes are cut in a selected piece of metal and control items such as buttons are attached to this metal "plate". The plate serves as a way of associating these various elements together to achieve a particular purpose. Similarly, SAW uses the idea of a control panel "plate". Most plates have a conventional window appearance. Components attach to a plates and may have spatial content such as graphics and Windows controls or non-spatial, purely informational content such as data structures, variables, and procedures. Plates may also have attached to them other plates, either "panel mounted" directly to the parent plate and continuously visible or "pendant mounted" such that the plate is normally invisible until requested to "pop up". This permits the construction of hierarchical plate assemblies.

Changing Component Attributes

Single clicking on a component, such as the default plate, selects that component for subsequent operations. The component is given "handles" which allow the developer to alter the position and size of the component by dragging the handles to new locations.

Double clicking on the component produces an editor which allows the developer to change the non-spatial attributes of the component. Different components provide different specialized editors to alter and customize component attributes.

Editing a Plate

Figure 4 shows the "Plate Editor" that appears when the default plate is double clicked.

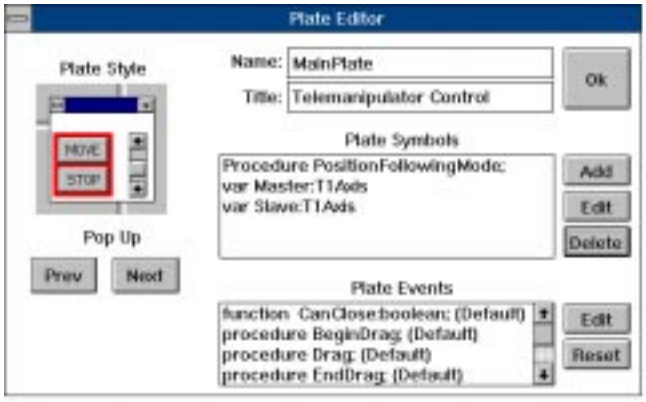

Figure 4. Plate Editor

Most components have names which are used to reference the component. The name is generally available to edit. In this plate editor the characteristics of the window associated with the plate are chosen on the left. A list of plate symbols is displayed on the right. Here the developer can specify procedures and variables that relate to the activity of the plate.

Event Response Methods

Below the list of plate symbols is a list of predefined plate events which have names such as "BeginDrag". These are event response procedures. SAW provides to the developer the popular event-driven user interface model. These event response procedures invoke when the corresponding events occur. BeginDrag, for example, invokes when the left mouse button is pressed and the cursor is on the window representing this plate. By editing the bodies of these procedures and including motion directives it is possible to respond to mouse activity and perform operations such as a "drag-and-drop" of the servo controlled mechanism.

Adding a Procedure

If the developer chooses to add a procedure an editor such as Figure 5 is provided.

Figure 5. Editing a Procedure

This particular editor is being used to write the force reflection algorithm.

Adding a Graphic Element

Components that have visual content are generally created by selecting the tool for that component and dragging a rectangle to indicate the component size and location. Figure 6 shows the addition of a chamfered bezel and indentation.

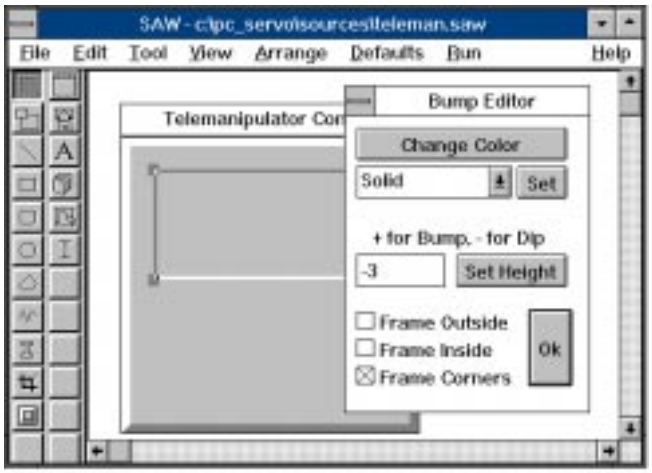

Figure 6. Bump Editor

These features are "raised surface" or "bump" graphics. The Bump Editor allows changing the depth, direction, framing, and color of the bump.

Displaying Information

Figure 7 shows the addition of several text components which provide labels and displays for application information.

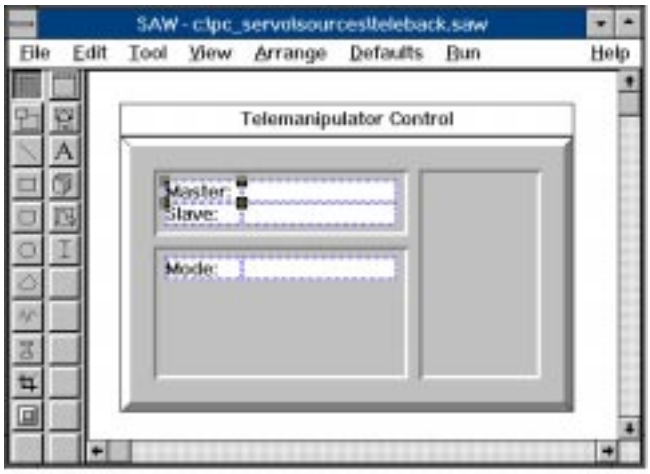

Figure 7. Adding Text Components

The text component for the master joint is named MasterDisplay and the slave's is named SlaveDisplay.

The procedure in Figure 8 displays the positions of the master and slave mechanisms in these text components using the Writeln method.

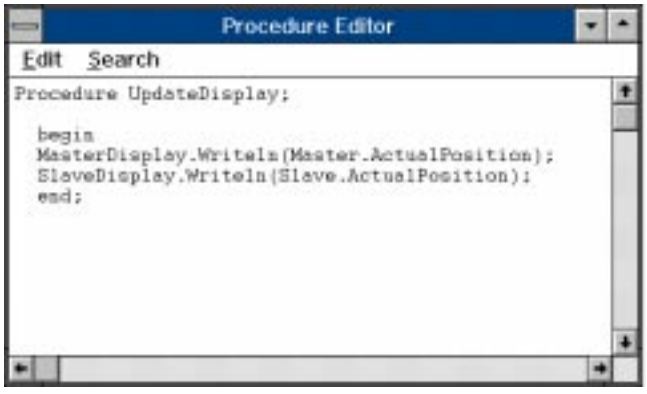

Figure 8. Procedure UpdateDisplay

Many objects that can receive information such as text, editors, or files, respond to Writeln. This UpdateDisplay procedure will be scheduled to run at 5 Hz by Setup, a plate event response procedure which invokes when the plate first begins operation.

Adding a Button

In Figure 9 the button that starts force reflection mode has been added and is being edited.

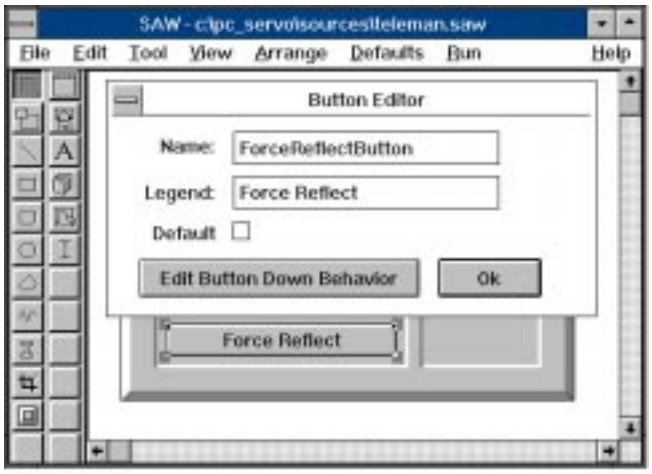

Figure 9. Button Editor

The button editor allows changing the name, legend, default status and "click" event response procedure for the button. Figure 10 shows the body of the click procedure. This procedure invokes when the button is pushed.

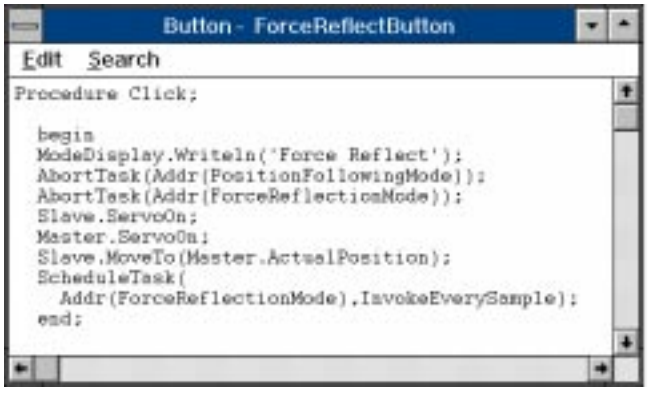

Figure 10. Force Reflect Button Click Procedure

The procedure updates the mode status in the ModeDisplay, aborts any previously started motion tasks, moves the slave to the master's current position, and schedules the force reflection task. Force reflection operates as an independent process at a frequency of 1 kHz, continually performing the multiple-input, multiple-output averaging control algorithm.

Adding a Catalog Component

One application requirement is to plot the position difference between the master and slave. The best way to provide data collection and display capability is to find a pre-fabricated subassembly which performs the desired operation. These pre-fabricated parts are accessed through "catalogs" invoked by choosing the catalog icon on the tool bar. Catalogs contain “data sheets” of component assemblies such as joysticks of various styles for various numbers of axis and storage scopes of various types.

The use of a catalog is very similar to the use of the Windows Help system. Summaries of components found in the catalog can be read by scrolling through the catalog index. The developer can also directly browse through the catalog using the forward and reverse browse buttons. A data sheet, such as shown in Figure 11, contains a title for the subassembly, a text description of what the subassembly is for and how it is used, and a picture of the subassembly.

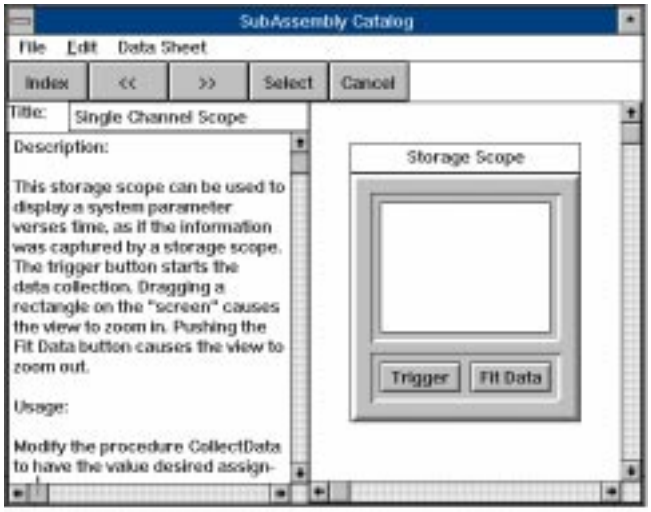

Figure 11. Subassembly Catalog

This particular data sheet describes a pop-up single channel storage scope with a "Trigger" button to initiate data collection and a “Fit” button that “zooms out” after “zooming in” to inspect detail. The subassembly is included in the application by pushing the select button.

The developer can create new data sheets for an existing catalog or create new catalogs by using “cut and paste” to move work from the drawing area into new data sheet pages.

The Hierarchical Package Metaphor

After selection the subassembly is placed into the application as a “package” as seen in Figure 12.

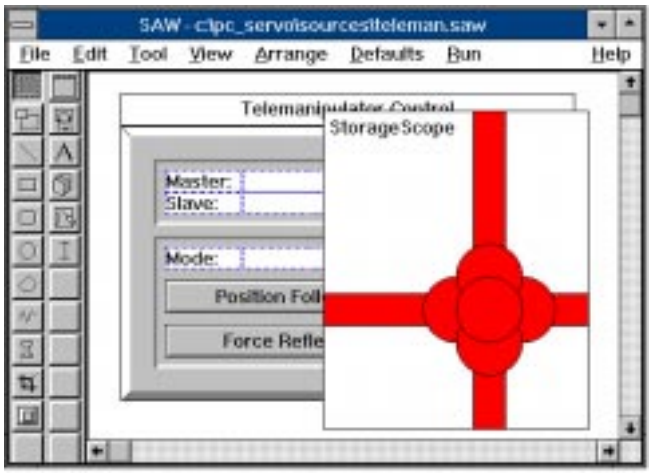

Figure 12. Gift Wrapped Subassembly

It is evident that the catalog component is a “package” because it has ribbons and a bow on it. Inside this “package” is the selected storage scope. The package conveys an important structural aspect of the Servo Application Workbench. Applications constructed with SAW are structurally hierarchical. Packaged components are subassemblies to the plate they attach to and distinct from other components.

The most conspicuous attribute of the storage scope is the appearance. However along with the appearance, inside the package, are the procedures, functions, objects, and variables which constitute the behavior of the storage scope. If a second storage scope is placed into the application it would appear as a second package. Even though it would contain copies of the same components (i.e. another trigger button and fit button) these components would be distinct from the first subassembly. The relationships of each component to other components in the package remains intact

regardless of the presence of duplicate component names being used in the application. Conflicts with other components having the same name are avoided because relationships are based first on what is found inside the package. The analogous software principle is “symbol scope”.

This isolation allows the developer to safely introduce new subassemblies into the application without duplicate name conflicts. Scoping is an important principle for any system providing conflict free component reuse.

Double clicking a package causes SAW to “open” the package and display the contents. The rest of the application is erased since it is no longer visible in the current scope. To see the run time appearance of the application uncheck the gift wrap Subassemblies display option in the SAW view menu.

The storage scope is connected to the error information by changing a statement in the storage scopes's data collection procedure as indicated by the storage scope's data sheet.

Figure 13 shows the addition of a button to invoke the storage scope.

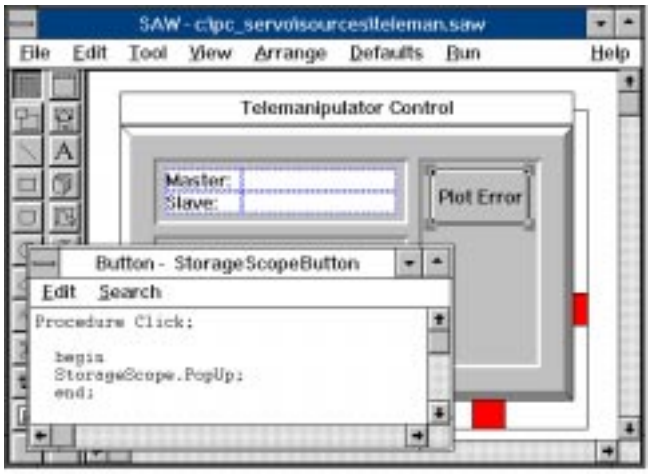

Figure 13. Popping Up a Subassembly

The click procedure for this button has one statement which directs the storage scope to pop up. Once popped up the controls on the storage scope are available to perform the data collection and display.

By using similar techniques additional controls are provided and additional functionality described to complete the application. Selecting the "Start" item from the "Run" menu in SAW causes the application to begin operation and have the appearance shown in Figure 1.

Development Environment Contributions

The following attributes help programs such as the Servo Application Workbench contribute to motion application development.

The Clip Art Metaphor

A good way to save time developing anything is to use high level components rather than primitives. When components have visual content browsing through a set of components aids in component selection. A metaphor for this process is “clip art”.

“Clip art” is published in books that can be purchased from the bookstore. Each book has a particular theme, such as “restaurant pictures” or “sports activities”. The pages of the book contain pictures of various sizes and types that relate to the theme of the book. By copying the page in the book you can “clip out” the desired image and paste it into a flyer or notice enhancing the flyer with an image that would have been very hard to draw. Clip art, although simple, is a powerful tool for the following reasons:

- Developers can conveniently access the work of others.

- The developer can start work at the component integration step rather than the blank-sheet-of-paper (or blank screen) step.

• Developers can recognize components in the catalog they might not know how to describe. This changes the development emphasis from synthesis to recognition.

• Developers can alter an "almost right" component found in the catalog and avoid generating the desired component completely from scratch. Even if exactly what is needed is not found, something close most likely can be found providing a significant head start.

• Catalogs invite browsing and study allowing the developer to learn from the structure and example of components that solve similar problems.

The catalogs of the Servo Application Workbench support the clip art metaphor and provide the application developer with these benefits.

"WYSIWYG"

What-You-See-Is-What-You-Get (WYSIWYG) is a powerful approach in constructing applications that have visual components such as controls and buttons. It is much simpler to show the system what is desired rather than describing numerically or symbolically a spatial component. Microsoft Windows provides a good environment for constructing WYSIWYG types of systems.

The Non-Preemptive Windows Problem

While constructing this application a procedure was written that scheduled the force reflection algorithm to execute as a separate, autonomous task at a frequency of 1 kHz. This was accomplished with a single program statement in the body of a fairly short procedure. However this represents a large contribution. Windows does not normally support this type of preemptive multitasking. SAW operates along with a behind-the-scenes Multitasking extension to Windows that allows this type of high frequency multitasking service to be provided in a simple and accessible way.

Any real time system that runs under Windows must either provide a solution to the Windows non-preemptive multitasking problem or be willing to "lock out" Windows operation for extended periods while the real time application controls the computer. The second "lock out" solution is usually not useful for interactive systems.

Windows is Difficult to Program

Windows is difficult to program regardless of the additional complications of real time applications. In the last year a new generation of language products has emerged that greatly simplifies the creation of Windows applications. Example programs in this class are Visual Basic and Turbo Pascal for Windows. These environments "hide" many of the complications of Windows programming allowing the application developer to focus on the problem at hand without being overwhelmed by Windows considerations, most of which the developer is not concerned with. The Servo Application Workbench is in this class of development environment.

Summary

Windows is fast becoming a familiar standard for operator interface systems. Without an application development framework, such as the Servo Application Workbench, motion application developers creating Windows applications are "on their own" to integrate together a high level language system, multitasking extensions to Windows, motion control system, Windows application development tools, and component reuse libraries. The benefit of the Servo Application Workbench is that the integration of these elements has already been accomplished. Access to the functionality of these components has been greatly simplified allowing the motion application developer to focus on the problem at hand rather than the complicated mechanics of a graphical user interface operating environment with real time extensions.

References

1) C. Petzold, *Programming Windows*, MicroSoft Press, Redmond, Washington, 1990

2) T. Swan, *Mastering Turbo Assembler*, Hayden Books, Carmel, Indianna, 1990

3) Intel Corporation, *i486 Microprocessor Programmer's Reference Manual*, Osborne McGraw-Hill Book Company, New York, 1990

About the Author

J. Randolph Andrews received his B.S. M.E. in 1981, B.S. E.E. in 1981 and M.S.M.E. in 1983 from the Massachusetts Institute of Technology. He participated in the MIT Mechanical Engineering Department's DeFlorez Design Competition each undergraduate year winning 1st place '78, 1st place '79, 1st place '80 and 1st place '81.

Andrews spent 4 years at Hewlett Packard's corporate research laboratory in the Applied Physics Research Center as well as the Manufacturing Research Center.

The following 4 year period was spent with Galil Motion Control.

In July '91 Andrews founded Douloi Automation to provide motion control hardware and software components primarily for use with Microsoft Windows.

Professional interests include motion control, software/electrical/mechanical system design trade-offs and high abstraction programming techniques and tools.