A Software Component Library for Motion Control

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

Paper presented at the
1993 Incremental Motion
Control Systems and
Devices Symposium

Introduction

As motion control applications advance in capability and sophistication it is necessary to package motion application behaviors in a manner that allows a developer to manipulate more and more abstract "building blocks" yet retain the flexibility to solve a unique problem with special requirements. Behaviors of interest to motion application developers include electronic cam operation, tangent servoing, and robot kinematics to name a few.

There are also needs for user interface behaviors such as joystick controls, motion renderings, curve editors and virtual instruments. These building blocks need to be so simple to use that they can be "dropped" into an application, yet accessible enough to be altered to meet a specific application need. The concept of building an application from standard building blocks is familiar for mechanical and electronic systems but has been elusive for the software system.

The objective of this talk is to provide an overview of what can now be done with software components as they relate to motion control applications. The following sections discuss Software Component Background, Component Mechanisms, Component Categories and Summary. Example applications are presented which demonstrate the benefits of developing with software components.

Software Component Background

Historically, the best way to make use of prefabricated software has been through "function libraries" which contain a collection of routines that relate to some particular purpose. Examples include matrix manipulations, graphic routines, or manipulating a particular add-on hardware card. These libraries and "drivers" have been a help to developers but have not had great impact. In most cases it was required that these libraries be written in the language that the developer has chosen to use. Because of the various languages available, and various implementations of any particular language, it was difficult to provide a broadly used library without a great deal of rewriting on the part of the library provider.

Software developers have long looked to computer hardware as a metaphor for a desired software component model. Hardware development has outpaced software development because of the ability of hardware engineers to readily use prefabricated components. This component reuse occurs on two levels.

Backplane Metaphor

A computer often has some "backplane" or interconnection mechanism into which high level circuit cards plug allowing them to communicate to each other. The backplane implements a set of communication standards and protocols that compatible cards follow. The emphasis of the backplane metaphor is communication between systems not necessarily made by the same vendor. Software developers have asked the question "why isn't there a software backplane that allows software components to communicate to each other in a similar manner?"

Integrated Circuit Metaphor

A second level of the hardware metaphor is the integrated circuit. Circuit boards are often composed of standard integrated circuits with well understood behaviors. In creating new circuit boards a hardware designer can pull off his shelf a data book containing a large collection of components and combine these components together to create a desired function. These parts also conform to interconnection standards. IC-to-IC interconnection usually operates at higher speed and performance than the backplane interconnection standard. These IC's represent a higher degree of functional granularity than the components that plug into the backplane. Through ASIC and FPGA technology, collections of standard cells can be combined together into a new integrated circuit. This integrated circuit metaphor is "nestable" allowing new integrated circuits to be composed from pre-existent integrated circuits. This has also been referred to as an "assembly" metaphor where new assemblies are composed of subassemblies. Software developers have asked the questions "Why isn't there a data book of software IC's that can be combined to create new functionality?".

Component Mechanisms

In the last several years, advances have occurred which to some degree provide the Software Backplane and Software IC to application developers. The following sections discuss the various ways this has occurred.

Windows Dynamic Link Libraries

Regardless of its technical merits, Microsoft Windows has become the graphical user interface of choice for the IBM PC. Although not normally a real-time environment, products such as Intel's iRMX for Windows and Douloi Automation's SI-3000 provide real time pre-emptive capabilities to Windows. The most useful software backplane advance has been through Microsoft Windows Dynamic Link Libraries. Dynamic Link Libraries communicate to applications and each other through a standard calling convention which is simple to create and simple to use. More importantly, dynamic link libraries written in different computer languages can interconnect with no additional consideration or difficulty. This is the key enabling step. Now a vendor can provide a single component that can be used by anyone regardless of the programming language they choose to use. Connecting to a DLL requires only that you describe the parameters to pass to the function and where the function is located. Some systems, such as Douloi Automation's SI-3000, can both respond to as well as request DLL calls from other software components.

Windows also provides Dynamic Data Exchange as an interconnection method. This is analogous to hooking up instruments through an IEEE-488, HPIB style communication link. Like such a hardware link, it involves resolving who is the "talker", and who is the "listener", and the protocol for the conversation. Although DDE is flexible and important, it is slower and more difficult to implement than Dynamic Link Libraries.

Object Technology

At the Software IC level, Object Technology has made important advances in creating encapsulated, reusable components. Although not a universal solution, object programming is an important tool in any developer's toolbox. Object Technology provides a helpful framework for organizing many types of problems. Objects can be created with languages such as C++, Borland Pascal, and Douloi Pascal. The major benefit of object programming is that it allows a developer to create new functionality which is similar to an already present functionality by specifying only the differences. Object technology tends to be very language specific.

Visual Programming

Visual programming is a term which has come to mean an interactive software development environment where the program is described spatially with drawing techniques as well as with the commands of a programming language. Examples of such programming environments include Microsoft's Visual Basic, Microsoft's Visual C++, National Instruments Labview, Digitalk's Part Workbench, Next Computer's Next Step, and Douloi Automation's Servo Application Workbench.

The level of abstraction of "primitives" in a visual programming system tends to be much higher than in a normal programming system. As well as conventional data types such systems also include more advanced components such as buttons and list boxes. The shape and position of these components are established by pointing with the mouse. The behavior of the component is modified through program commands and attribute settings.

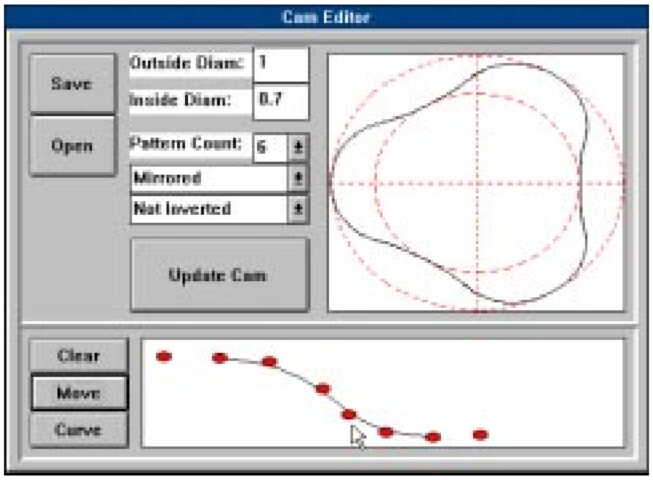
Component Categories

Software component technology is very important for software application development in general. What types of components or functions are most important for motion applications in particular? The following categories are frequently encountered. Along with a description of each category, some example applications are shown to illustrate the role software component technology has had on the application development.

Motion Coordination

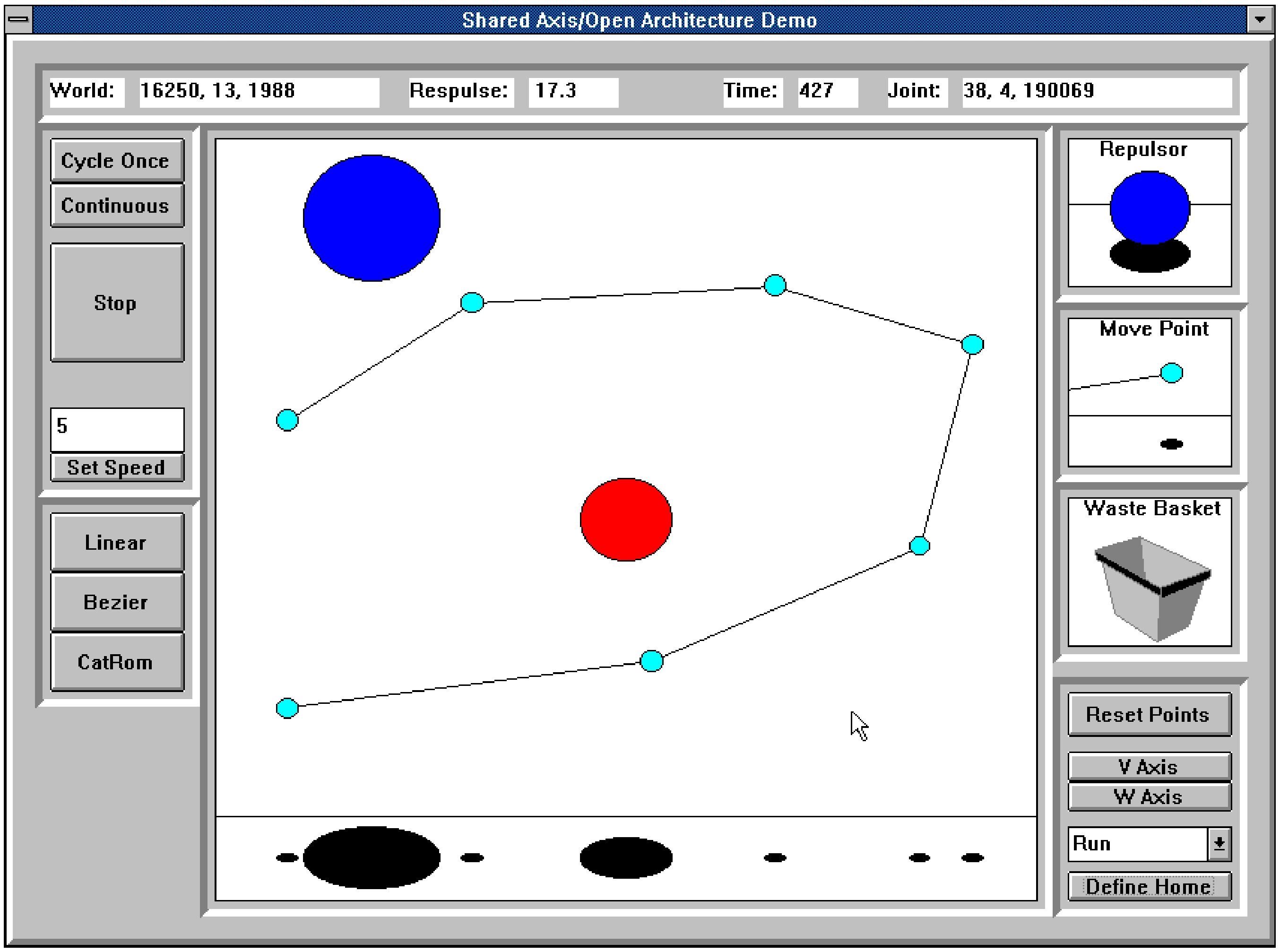
Motion coordination components are used to provide or support specialized relationships between axis such as electronic gearing, tangent servoing, moving-frame manipulation, and robot kinematics.

Figure 1 shows an “Electronic Cam Editor”. The basic shape of a cam lobe is portrayed as a curve in the lower drawing area. The curve is placed and altered by dragging the descriptive points with the mouse. The upper drawing area displays the resulting shape of the cam after the lower image has been wrapped around a cylinder a selectable number of



Outside Diam: 1
Inside Diam: 0.7
Pattern Count: 6
Mirrored
Not Inverted

Figure 1. Electronic Cam Editor



World: 16250, 13, 1988
Respulse: 17.3
Time: 427
Joint: 38, 4, 190069
5
Run

Figure 2. Axis Sharing/Obstacle Avoidance Motion Application

times, possibly mirrored back-to-back and possibly inverted. The resulting representation can then be used to prescribe the displacement of a mechanism based on the input angle of the cam in a master-slave manner. This component was created with Servo Application Workbench in about a day.

Figure 2 shows a motion application used to drive a spherical robot having 3 spatial degrees of freedom and a radial wrist rotation. The medium sized sphere in the middle of the work area represents the robot's tool location. Dragging and dropping the sphere on the computer screen causes the robot to reposition the tool in a corresponding vertical Cartesian plane. The small circles are dragged into position on the computer screen with the mouse to prescribe the Cartesian trajectory. The black ellipses below the circles are “shadows” helping to convey a three dimensional effect. The small spheres can be interpreted as control points to describe a curve from a number of available curve types or used as destinations for linear interpolation. The orientation of the tool held by the robot is prescribed by an external sensor and may be independent from the tool’s Cartesian position even though the rotation of the tool and movement of the tool kinematically share two axis. The largest sphere positioned on the screen represents a repulsive force field. This force field pushes away the Cartesian trajectory if it passes nearby providing a type of obstacle avoidance. What would have been a straight line in Cartesian space is changed into a Cartesian space curve which dodges the repulsive sphere.

Object technology helps represent the spheres on the screen and is involved in representing the robot. The different elements of this application, axis sharing to provide independent tool rotation, Cartesian-to-spherical kinematics, and Cartesian space repulsive force field obstacle avoidance, are calculated and combined at 1 kHz to provide commanded setpoint information to the servos at the controller sample rate of 1 kHz While this is taking place the user is able to direct the activity of the system by dragging and dropping the different user interface elements with the mouse.

The use of repulsive force fields to deflect a robot trajectory was explored in a Master’s thesis written in 1983 and required many months to implement. The benefits of software component technology allowed this application to be constructed in approximately 3 days with Servo Application Workbench.

Joysticks

Joystick components give the operator of a motion control system the ability to manually direct the movement of the mechanism during a training phase, or as part of the setup of the machine. Some joystick components relate to switch closures of physical hardware and interpret a closed switch as a request to perform motion at a constant speed. Using the computer mouse it is possible to “drag and drop” a mechanism as was described in the spherical robot example. It is also possible to create “virtual joysticks” which appear on the computer screen and have the appearance of a physical

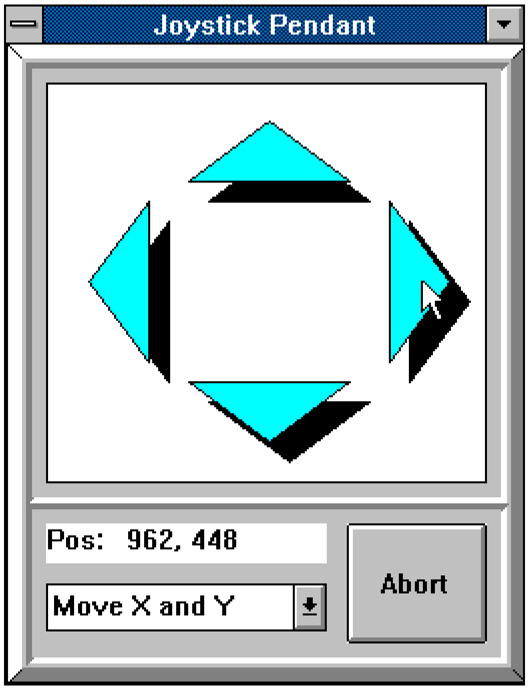


Figure 3. Virtual Joystick

joystick. Figure 3 shows an example of a 2 dimensional virtual joystick.

The joystick “pops up” when requested and can be moved around on the screen so as to not obscure information which might be beneath it. Clicking the mouse on one of the arrows causes the mechanism to move in the indicated direction. The further away from the center of the joystick, the faster the motion. Letting go of the mouse button causes the mechanism to come to a stop. Experience to date has shown that “position” joysticks, such as in the previous example, are much more useful than “velocity” joysticks such as this one although usefulness varies from application to application. This joystick was constructed in about 30 minutes.

Displays

Display components present information or state about the system to the user. Information might be textual or graphical. Graphical information might be expressed as a chart, or as the positions and shapes of geometric items. Figure 4 shows a small "spin gauge" alongside a position display.

This display component is used to show the position of the X axis motor. When the X axis is commanded to move, the gauge spins and the position value changes indicating the movement. This is helpful for software development in the absence of a mechanism or during a demonstration.

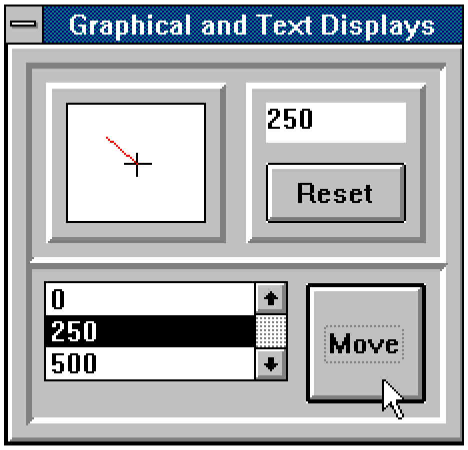
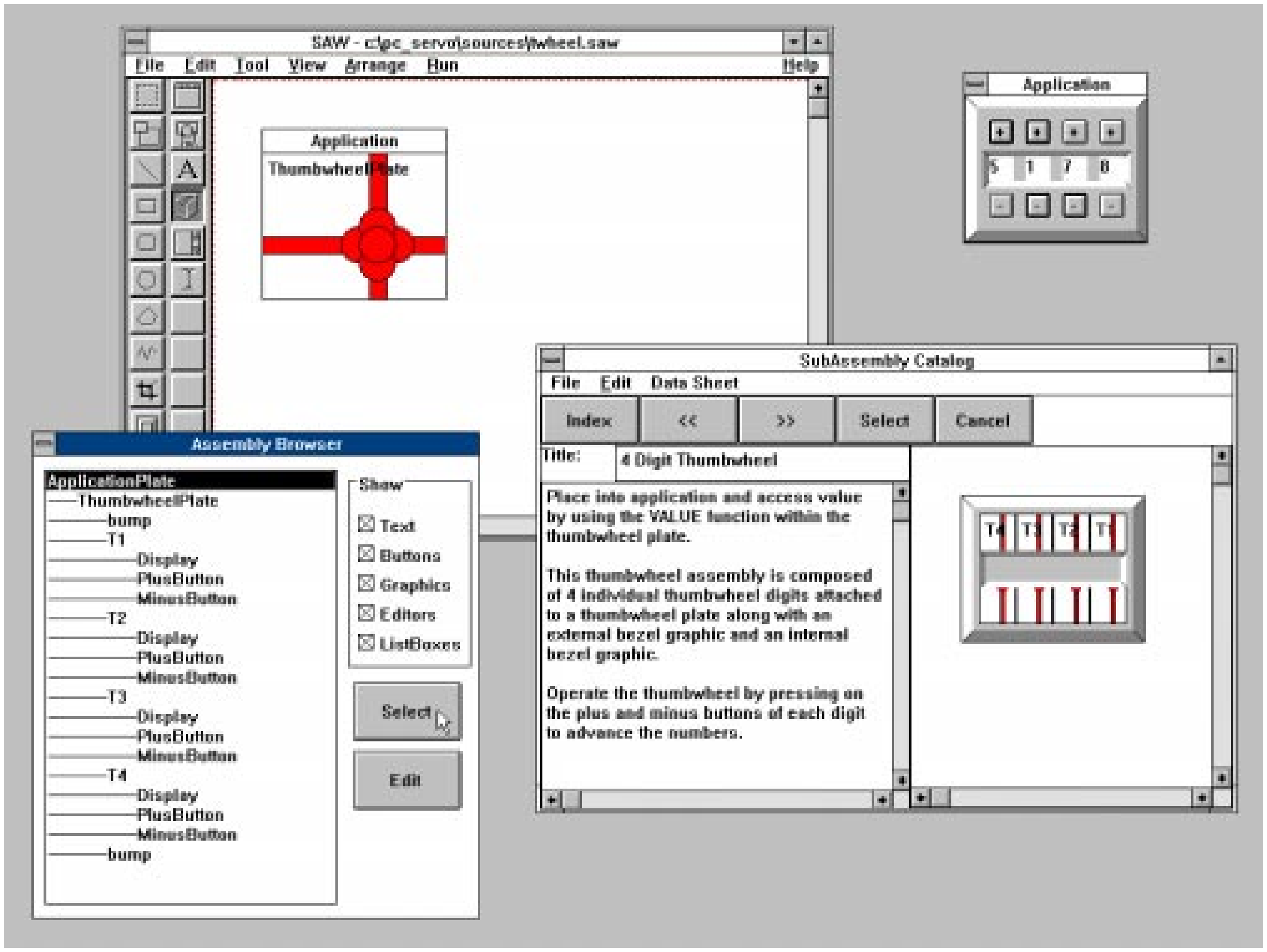


Figure 4. Graphical and Textual Displays

Controls

Control components effect or establish a new state for the motion application. Controls can include components such as buttons, list boxes, thumbwheels and numeric editors.



```
ApplicationPlate
 ThumbwheelPlate
   bump
   T1
     Display
     PlusButton
     MinusButton
   T2
     Display
     PlusButton
     MinusButton
   T3
     Display
     PlusButton
     MinusButton
   T4
     Display
     PlusButton
     MinusButton
   bump
```

Title: 4 Digit Thumbwheel

Place into application and access value by using the VALUE function within the thumbwheel plate.

This thumbwheel assembly is composed of 4 individual thumbwheel digits attached to a thumbwheel plate along with an external bezel graphic and an internal bezel graphic.

Operate the thumbwheel by pressing on the plus and minus buttons of each digit to advance the numbers.

Figure 5. Thumbwheel Control with Subassembly Catalog and Assembly Browser

Figure 5 shows a page from a Thumbwheel Subassembly Catalog in Servo Application Workbench. The “virtual” thumbwheels in the software catalog have a very similar structure to their mechanical counterparts. The Assembly Browser shows that the thumbwheel is composed of a number of individual digit modules, and that each module is composed of a display, a plus button, and a minus button. These components are butted up against one another and then grouped with a “bezel” graphic much like a mechanical thumbwheel set would be. This thumbwheel is composed of subassemblies, and itself becomes a subassembly to the plate that it is attached to in the application. This illustrates the assembly metaphor frequently encountered in visual programming. This thumbwheel catalog, containing the digit module, 6 arrangements of thumbwheels, and an example application, was built in about an hour.

Save/Restore

Save and Restore components are used to preserve the current configuration of the system from one invocation to another. Configuration information includes compensation and profile parameters. It is also possible to save and restore vector arrays which can be used for storing taught positions.

Instruments

Instruments are higher level components which assist in specific tasks usually related to setup or diagnostics. Examples of instruments include storage scopes, input monitors and output switch pendants. Instruments are usually composed of display and control components and mimic physical instruments to convey familiarity to the user.

Figure 6 shows a series of instruments used to test servo control hardware. The Analog Test instrument is used to set the open loop torque on the hardware and can be used for setting the offset. The Output Test instrument manipulates output bits to confirm proper operation. The Check Encoders instrument displays encoder positions for each of six axis and

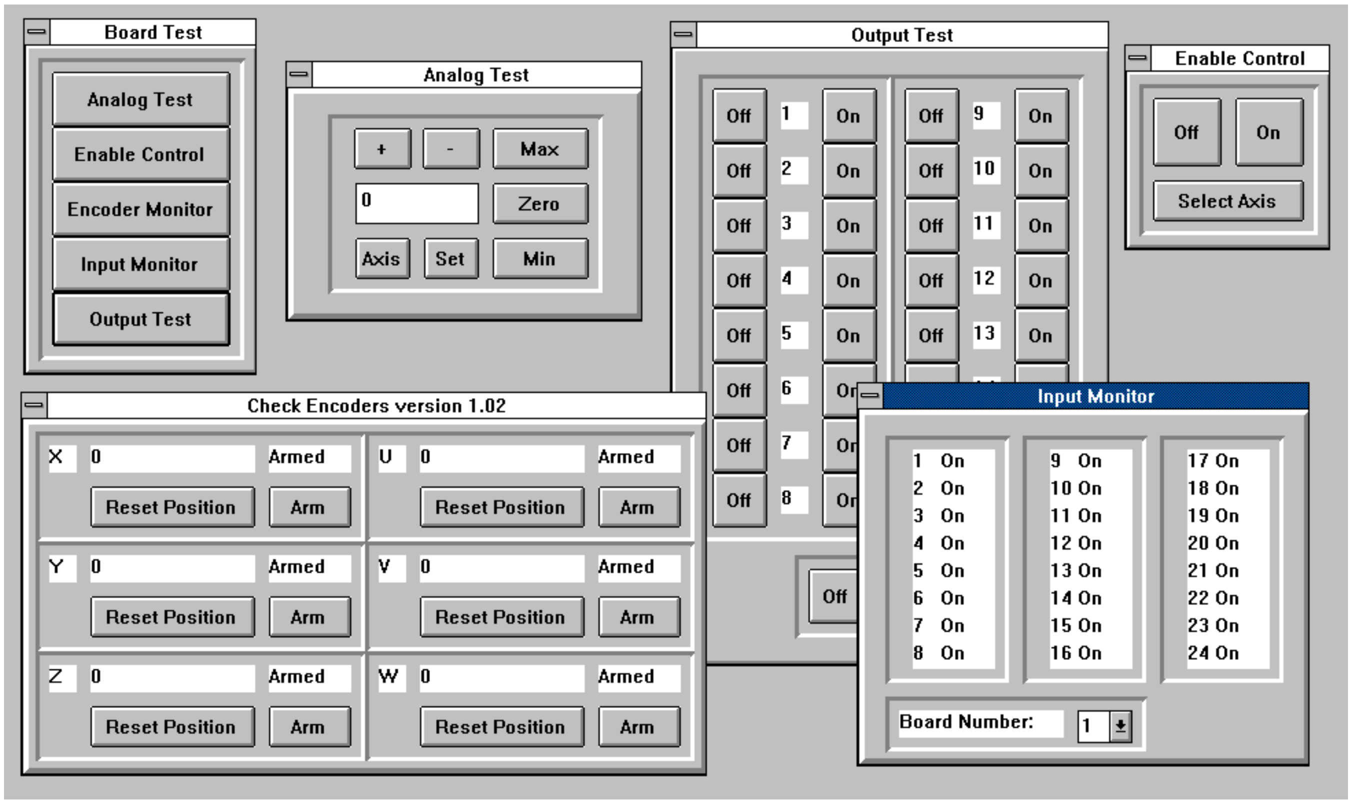


Figure 6. Virtual Instruments used to Test Servo Control Board Hardware

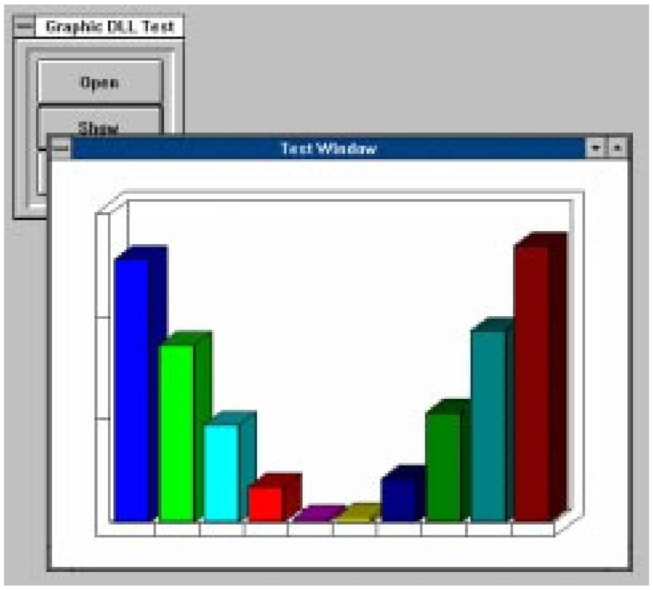


Figure 7. Graphics DLL Connected to Application

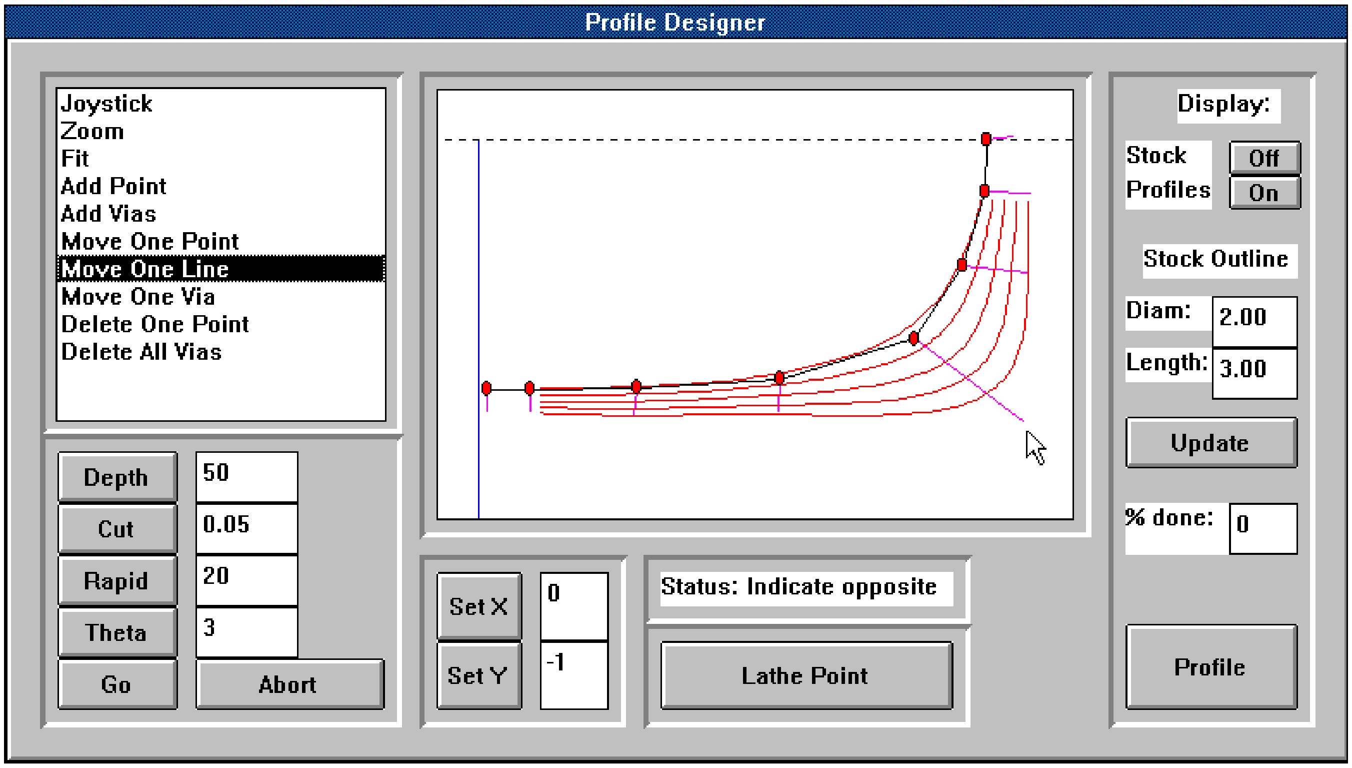
allows for the arming and triggering of capture inputs. The Input Monitor presents the states of the input ports. The Enable Control manipulates amp enable signals to check proper operation.

Connectors

Connector components act as "patch cords" between software components or between a software component and a hardware component. Software-to-hardware connectors serve as "drivers" for the hardware and present the functions of the hardware to the development environment in more abstract and simple to use terms. Software-to-software connectors usually bridge the gap between a language specific object environment and a language independent software backplane. Figure 7 shows a 3 dimensional bar chart representing information collected from an application. The chart appears in response to pushing a button on the motion application control panel and behaves as if it was part of the application however it is not. Charting capabilities are provided by a third party Windows DLL. The application requests charting services on its behalf by communicating to the DLL through the software backplane. The connector component which allows the motion application to make these requests provides a description of the procedures and functions available and indicates the name of the DLL where they are found.

Function Libraries

Although function libraries are the oldest packaging method for software components such libraries are still



Depth 50
Cut 0.05
Rapid 20
Theta 3
Set X 0
Set Y -1
Status: Indicate opposite
Diam: 2.00
Length: 3.00
% done: 0

Figure 8. Curve Interpolations Perfomed by DLL Library

useful. Function libraries helpful to motion control applications include curve fit libraries and string manipulation. Figure 8 shows a turning lathe control panel where the desired part shape has been expressed as a series of points. Curve fit techniques are used to describe the desired part shape as well as a series of intermediate shapes needed to change the raw stock into the final part.

Summary

A mechanical engineer would be considered foolish to specify and fabricate a standard screw if it could be purchased. The realization of prefabricated components has moved through the mechanical design mainstream and the electrical hardware design mainstream. At this point in time software component technology is just beginning to move into the software design mainstream in the form of Windows Dynamic Link Libraries, Object Technology, and Visual Programming.

The additional complexities of developing a motion application require all of the help that can be brought to bear by these techniques. Benefits of software components include an improved user interface, more reliable operation, and more rapid application development.

Bibliography

Andrews, J. Randolph: "A Servo Application Development Environment for Microsoft Windows", In *Proceedings of the Twenty First Annual Symposium on Incremental Motion Control Systems and Devices, San Jose, CA*, 1992.

Andrews, J. Randolph: "An Advanced Motion Control System Architecture Based , On a 386 PC", In *Proceedings of the Twenty First Annual Symposium on Incremental Motion Control Systems and Devices, San Jose, CA*, 1992.

Andrews, J. Randolph: *User Manual for Servo Application Workbench.* Douloi Automation, 1992.

Andrews, J. Randolph: "Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator", Master's Thesis, MIT Dept. of Mechanical Engineering, 1983.

Brogan, William: *Modern Control Theory*, Prentice Hall, New Jersey, 1991

Budd, Timothy: *An Introduction to Object-Oriented Programming* Addison-Wesley, Massachusetts, 1991

Coad, Peter & Yourdon, Edward: *Object-Oriented Analysis*. Yourdon Press, New Jersey, 1990

Cox, Brad: *Object Oriented Programming: An Evolutionary Approach*, Addison-Wesley Publishing, 1986, 1991

Ellis, George: *Control System Design Guide*, Academic Press, San Diego, 1991

Franklin, Gene & Powell, David: *Digital Control of Dynamic Systems*, Addison-Wesley, Massachusetts, 1981

Liskov, Barbara, & Guttag, John: *Abstraction and Specification in Program Development*, MIT Press. Massachusetts, 1986

Meyer, Bertrand: *Object-oriented Software Construction*, Prentice Hall, New York, 1988

Peters, Lawrence: *Advanced Structured Analysis and Design*, Prentice-Hall, New Jersey, 1987

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 and 2nd 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 for use with Microsoft Windows.

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