ACCELERATOR CONTROL SOFTWARE CONSTRUCTION BASED ON SOFTWARE OBJECT COMPONENTS[*]

C. Timossi, H. Nishimura, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

Abstract

We will be presenting the results of a recent effort on the use of software object components for software construction for the Advanced Light Source control system [1]. Components are written for Win32 for low-level access both to the current control system and to EPICS [2] Channel Access, and for higher level physics tools. We will discuss the merit of the component-based approach based on our experience with several examples.

1 BUILDING ACCELERATOR APPLICATIONS

Much work has been done at the ALS to provide object oriented class libraries for both control of the machine and for modeling and simulation of machine behavior [3]. As new devices are added to the accelerator and the current control system begins it's migration to EPICS, some weaknesses of continuing to build applications, especially new graphical applications, using these libraries has become apparent.

There is a trend in the industry, both in operating system and application design, towards component based architectures such as Microsoft's Component Object Model [4] and Sun's JavaBeans [5]. We feel that there are clear benefits in following this trend for the construction of accelerator applications. As a test, we built new components and migrated parts of existing class libraries to components using them to build several applications.

2 COMPONENT SOFTWARE

The goal of using component based software construction is to enable rapid development of applications, especially graphical applications, from pre-assembled components while allowing the developer the broadest choice in development tools. A *component* is distinguished from a subroutine or class library, in the way it makes itself available to its client (container). A component can make much more information about itself available at run time in a language independent way. The exact way that this information is published is dependent on the component architecture, but usually involves some process of registration with the operating system. The type of information that becomes available to the client is the component's methods and method signature (parameters), its attributes, and which types of events it can handle (sink) and can generate (source). The creation of such a component is more complicated than the creation of a subroutine or class library due to the need to conform to the architecture. What is gained from the effort is the ability to build applications using a wide variety of existing development tools that conform to the architecture. The containing application will have full access to the component and may also let itself be driven from the component.

2.1 Win32 components

Though component standards such as Sun's JavaBeans promise platform independent components, we restrict ourselves to the Microsoft Win32 platforms (Windows NT and Windows 95) for two reasons. First, it is the dominant platform at the ALS and second, there are wide variety of development tools for building components and containers.

Components on Win32 platforms conform to the Component Object Model (COM). The process of component registration follows these steps: a component's interface is specified using an Object Definition Language (ODL) in which each interface is tagged with a globally unique identifier (GUID), then the ODL is compiled to produce a *type library*, which, along with the executable for the component, is registered with the operating system. Once registered, a variety of tools can view the component's interface.

2.2 ActiveX Controls

An ActiveX *control* is a type of a COM component that implements a defined set of interfaces. The applications or development tools that use them are called *containers*. We find that these controls are the most useful in application construction because of the many third-party software packages that serve as Active X containers. In particular, we are using Borland's Delphi 2 and Microsoft Visual BASIC 5 development tools for building applications that contain our controls.

Controls can have rich graphic content, such as the one developed to display a tune-plot, or they have just be invisible helper controls such as the ones implemented to access accelerator data.

A control has the ability to serve as a 'source' or 'sink' of events, either system events, such as mouse movement, or user defined events.

[*] This work was supported by the Director, Office of Energy Research, Office of Basic Energy Sciences, Material Sciences Division, U. S. Department of Energy, under Contract No. DE-AC03-76SF00098

3 EPICS CHANNEL ACCESS CONTROL

Since a major source of accelerator data is generated by control system processors running the EPICS software, a low level component that handled channel access (CA), the network protocol for EPICS, was an obvious first choice.

3.1 Channel Access monitors

A channel access client application can ask a server to send updated data for a particular *process variable* (PV) whenever that variable changes by a significant, user defined, amount. This mechanism is called placing a monitor on a variable. We wanted a control that would allow the user to enter a PV name, would monitor the PV, and would fire events to the container when a new value was detected.

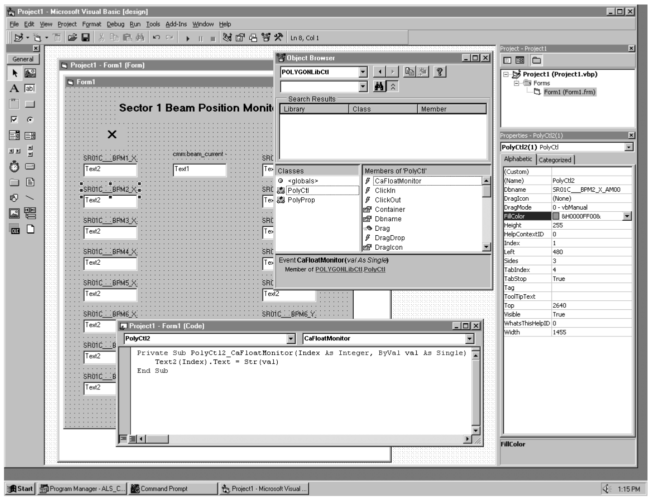
3.2 Control architecture

The control was built with Visual C++ and the Active X Template Library (ATL), a Library of C++ templates for implementing COM objects with minimal overhead. The control is linked to a Dynamic Link Library, ca.dll, that we ported for Win32 to handle the channel access protocol.

When the control is placed on the container, a property sheet is used to enter the name of the PV to be monitored. When the first PV monitor object is created, a *pend thread* is created to listen for monitors and to call the object's handler when a change occurs in the PV. The handler then fires a *CaFloatMonitorEvent* to the container which will usually handle it by updating some graphic element. Since ca.dll is not thread-safe, a mutex semaphore is used to avoid conflicts between the pend thread and other CA routines.

3.3 Beam Position Monitor application

An application was built using Visual Basic 5 that displays the digital values of 16 of the BPMs in one section of the ALS Storage Ring. In addition, a third party control, Pinnacle Publishing's Graphics Server, was used to display a line plot of the data. This application illustrates the usefullness of combining controls from various sources into a final application.



```
Private Sub PolyCtl2_CaFloatMonitor(Index As Integer, ByVal val As Single)
    Text2(Index).Text = Str(val)
End Sub
```

(Custom)	
(Name)	PolyCtl2
Dbname	SR01C___BPM2_X_AM00
DragIcon	(None)
DragMode	0 - vbManual
FillColor	&H0000FF00&
Height	255
HelpContextID	0
Index	1
Left	480
Sides	3
TabIndex	4
TabStop	True
Tag	
ToolTipText	
Top	2640
Visible	True
WhatsThisHelpID	0
Width	1455

Fig 1. BPM display construction with Visual Basic

4 ACCELERATOR PHYSICS CONTROLS

4.1 Existing Class Libraries

There are two class libraries in wide use at the ALS: Goemon and TracyLiB for modeling and DMM96 [3] for online control. These libraries were implemented in both C++ and Object Pascal/Delphi. Moving their functionality to an ActiveX Control, written in Microsoft Visual C++, allows them to be maintained in one language and still be used by applications written in either language.

4.2 Modeling and Simulation

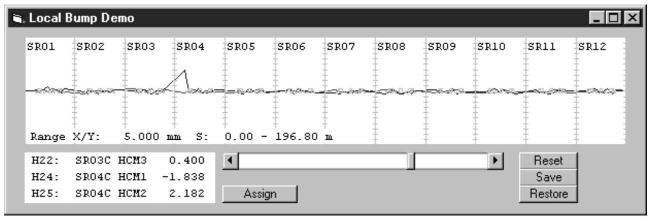
For the development of TracyV [6], a visual and interactive machine simulator for the ALS, a version of Goemon, called TracyLib, was developed in Delphi2 to ease graphic application development. ActiveX Controls have been developed to implement the more flexible Goemon and yield the graphics capability of TracyLib that can be used in both Delphi2 and Visual C applications.

4.2.1 Tune Diagram and BPM Display Controls

There is a set of graphs, such as tune diagrams, beta and dispersion function diagrams and dynamic aperature display, commonly used in modeling and simulation. ActiveX controls have been implemented for use at the ALS that show a tune diagram and a BPM display, that can be used directly in an active X container. Fig. 2 shows an example of a local bump emulation assembled on a Visual Basic form.

4.2.2 Smatrix and Local Bump Controls

An Smatrix-based method for orbit control is in use at the ALS as described in a previous paper [7]. The Smatrix component encapsulates a sensitivity matrix and its manipulation routines, that can be used for both orbit control and machine diagnostics and characterization studies.



```
Range X/Y:    5.000 mm   S:   0.00 - 196.80 m

H22:   SR03C HCM3    0.400
H24:   SR04C HCM1   -1.838
H25:   SR04C HCM2    2.182
```

Fig.2. BPM display control showing a local bump emulation.

The local bump component implements local orbit bump using three steering magnets. This component is evolving into a general orbit control component by implementing other algorithms such as the most-effective corrector method and the singular-value decomposition method.

4.3 Machine Control

A subset of the DMM96 library is being supported as a collection of ActiveX controls built using MFC. Instead of mirroring the class structure of the existing C++ class library, components have been implemented that ‘wrap’ a group of classes that are used by application programs.

5 CONCLUSIONS

Components, such as ActiveX controls for Win32 systems, are still difficult to construct even with the new tools available. Once they are constructed, however, they allow efficient and rapid graphical application development using very different environments, such as Delph2 and Visual Basic 5.

REFERENCES

[1] S. Magyary, et al.,”The Advanced Light Source Control System”,Nuclear Instruments and Methods in Physics Research A293 (1990) p36-43
[2] L.R. Dalesio, et al.,”EPICS Architecture”, ICALEPCS 91, KEK Proceedings 92-15, (1992)pp.278-282., Dalesio,L., et al. “The Experimental Physics and Industrial Control System Architecture, ”,ICALEPCS,Berlin, Germany,Oct.18-22,1993.
[3] H. Nishimura, “Taking an Object-Oriented View of Accelerators,” IEEE 95PAC, 95CB35843(1996)2162
[4] Microsoft Corporation and Digital Equipment Corporation, “The Component Object Model Specification”, Draft Version 0.9, October 24, 1995
[5] Sun Microsystems, “JavaBeans 1.0”, Dec. 4, 1996
[6] R. Keller, H. Nishimura, et al.,”Orbit Stability of the ALS Storage Ring”, 97PAC
[7] H. Nishimura, L. Schachinger and H. Ogaki, “Orbit Control at the ALS Based on Sensitivity Matrices”,IEE 95PAC, 95CB35843(1996)2247