HERA CONSOLE APPLICATIONS BASED ON ACOP

Philip Duval and Honggong Wu, DESY, Hamburg, Germany

Abstract

The ACOP (Accelerator Component Oriented Programming) ActiveX control is now used in a wide variety of console applications in the control of HERA and its pre-accelerators. It has proven itself to be a versatile graphics control in its own right, with an intuitive interface for the developer. Its primary function, however, has been to provide a common Application Programmer's Interface (API) for three rather different data exchange mechanisms found in accelerator control at DESY. These include the TINE data exchange protocol, MKI3 data exchange, and Channel Access. At DESY, ACOP is primarily used in console applications programmed in Visual Basic 5.0 running on Windows NT. Nevertheless, there are a number of applications which use ACOP in MS Visual C++, Excel and HP VEE for example. In all cases, the ease in programming in a high-level language such as Visual Basic with components has proven to be a marked advantage. We report here on our first year's experience using ACOP at DESY.

1 INTRODUCTION

The ACOP[1] ActiveX control is based on the Microsoft™ COM (Component Oriented Model) foundation, which implies both binary reusability and dynamic linking with its container. Just as important, it offers a language-independent interface. The most popular ActiveX control containers are typically Visual Basic and other Visual Basic-based applications such as Excel, although integration in Visual C++ and Visual J++ for instance is also straightforward. Some other Win32 applications, such as LabView and HP VEE, can also act as ActiveX containers.

Primarily, ACOP.OCX is a client-side component and offers an interface for accessing front-end devices. Secondly, the object offers a control system oriented graphics package for on-line data analyses. This package has been streamlined for efficient data-rendition at high frequency in a number of styles of paramount interest to the client-side applications running in the control room.

The ability to include a component with a well-known interface inside the development application of choice has great advantages over integrating say a class library written in C++. The latter can of course be accomplished. However, there are typically different integration strategies among target applications (e.g. the interface to methods and properties in LabView will be accomplished differently than in Visual Basic), and the interface might turn out to be language specific (i.e. only works in C++ unless a C-wrapper exists) or require a knowledge of C++. A COM component on the other land is language neutral and can be immediately plugged into any application that supports ActiveX.

In the ActiveX world, components are "self-contained, programmable, reusable, language-neutral pieces of code easily pluggable into applications." [2] Furthermore, components expose properties and methods, fire events, and are unequivocally identified through name and/or ID.

The ACOP strategy violates the "self-contained" ansatz to a certain extent, in that the interface to ACOP.DLL is required. ACOP.DLL itself is likely to require secondary DLLs in order to communicate with a control system(s). However, if installed properly, this "violation" is well hidden from the user.

2 ACOP.OCX

ACOP.OCX is the binary ActiveX control, which can be downloaded and installed as is. It provides a client-side component interface divided into two categories: data acquisition and data rendition. The interface to ACOP.DLL is largely concerned with data acquisition and highlights the facility-dependent development which must take place in order to benefit from ACOP. The data rendition aspect of ACOP is self-contained within ACOP.OCX to the extent that control-system specific format types are not used in the drawing methods (If they are, then ACOP.DLL will be queried for pertinent information).

The developer interfaces to ACOP by setting properties of, calling methods of, and responding to events of ACOP.OCX. We discuss the data acquisition and data rendition aspects separately below.

2.1 Data Acquisition

The settable data acquisition properties **DeviceName**, **DeviceProperty**, **AccessRate**, **AccessMode**, and **AccessProtocol** are all passed through ACOP.OCX to ACOP.DLL as non-fixed length strings. So any string parsing or interpretation

that takes place, occurs in ACOP.DLL and reflects the control-system dependent implementations there. The **AccessProtocol** property is designed to specify the control system protocol. At DESY, ACOP provides a common layer for the TINE protocol [3], the MKI3 protocol [4], and Channel Access [5]. Therefore, this parameter can be "TINE", "MKI3", "CA", or "AUTO_DETECT". The **AccessRate**, although passed as a string, is converted to an unsigned long value representing milliseconds in the DESY implementation. This value also serves as a timeout parameter for simple READ or WRITE requests. **AccessMode** is used to specify the nature of the data acquisition (at DESY: "READ", "WRITE", "POLL", "REFRESH", "RECEIVE"). The **DeviceName** property follows the naming convention:

[//<facility>/]<device server>/<device>

For instance "//HERA/BPM/WL197" or simply "BPM/WL197" (using the default facility) targets device WL197 from the BPM sub-system. **DeviceProperty** follows TINE conventions and reflects the property to be requested from the device in question. Channel access names can be specified either as one long string in **DeviceName** or split between **DeviceName** and **DeviceProperty** as per TINE convention.

The remaining settable data acquisition property Grouped is a boolean value which governs the behavior of event notification when more than one data link is bound to a particular control.

The above provides a comfortable set of data acquisition properties which can completely specify the data request. To be sure, one could argue for taking a narrower or wider approach. However, most object-based or object-oriented control systems make use of the above elemental descriptions in making a data request.

Read-only data acquisition properties include **Status** (a string value containing the data acquisition status) and **Timestamp** (the unsigned long UNIX timestamp associated with the data acquisition).

The principal data acquisition methods are **Execute**() (synchronous) and **OpenLink**() and **AttachLink**() (asynchronous). All three of these methods utilize two (optional) data sets, a primary set and an extended set, so that the argument list takes the form of:

[data,] [len,] [type,] [dataEx,] [lenEx,] [typeEx].

All arguments are optional, which means that they are passed to ACOP.OCX as OLE Variants. This makes life very easy for Visual Basic Applications, where an optional argument is simply omitted. The difference between the two asynchronous calls **OpenLink**() and **AttachLink**() is that the latter binds (attaches) the remote data to the addresses passed in [data] and [dataEx]. Thus these parameters must be globally available. **OpenLink**() on the other hand does no such binding. Therefore, when an event notification arrives, it is necessary to use the secondary call **GetData**(data, [dataEx], [hLink]) to actually obtain the acquired data into the application. At DESY, the extended data set [DataEx] quite generally refers to input data sent to the front-end server, where as the primary data set [Data] refers to the output data, returned from the front-end server. We note here another feature of ACOP, namely that if the data elements [Data] or [DataEx] represent fundamental OLE data types (such as arrays of shorts, longs, floats, or doubles) then the parameters [len] and [type] may be safely omitted. In such cases, the content of the elements will be scanned to determine the array length and the data type. If special user-defined data types are used, then these secondary parameters must be supplied, in which case ACOP.DLL is queried as to the nature of the data type.

As intimated above, the asynchronous calls fire an event when incoming data arrive at the client-side. The ACOP event that is fired in this case is the **Receive**(hLink, StatusCode) event. When this event is fired, the client program can check the StatusCode for success and the link handle in case more than one data links are bound to the same control.

ACOP.OCX passes the data acquisition parameters blindly to ACOP.DLL, whose duty it is to redirect the request to the appropriate control system.

2.2 Data Rendition

The data rendition aspects of ACOP are indeed self-contained. The idea is to offer a simple set of display tools for the most common categories of data display. This includes for instance displaying data arrays as traces or histograms and offering various zooming and scrolling possibilities. We shall not go into any great detail here as the details are many, but instead refer the reader to the ACOP documentation from the ACOP web page. We note only that, once again, a good many of the parameters in the display methods are optional and in decreasing order of frequency of use.

3 ACOP.DLL

In order for ACOP.OCX to work properly, ACOP.DLL must exist on the path, and export the following interface routines:

void WINAPI **InitAcopDll**()
long WINAPI **DevRequest**()
long WINAPI **QueryDefaultAccessProtocol**()

long WINAPI **QueryDataBlockSize**()
long WINAPI **QueryStockAccessModes**()

We shall not go into the details of the calling arguments here. Suffice it to say, that such details are covered in the ACOP documentation, and that a sample ACOP.DLL with source code is provided in the ACOP download package. It is then a matter of adapting the sample DLL to the control system protocol(s) in question.

4 CONTAINERS

The principal development language for console control applications at HERA is Visual Basic. ACOP has been seen throughout the past year to be wonderfully suited to this environment. Useful control applications can literally be written in a matter of minutes. This is also true of applications which use VBA (Visual Basic for Applications) as a macro language, such as Excel. High-level development environments such as HP-VEE™ and LabView™ can also easily incorporate ACOP, which has been a considerable help to control engineers.

Microsoft Visual C++ (MSVC) can likewise act as a container for ACOP, although here, its use is not as straightforward as in the previously mentioned cases. The reason for this is the use of OLE variants in most of the ACOP method calls. Whereas in the higher-level containers, type-casting into variants is automatic, and "optional" parameters can be omitted, in C++, OLE variants must be specifically allocated and assigned, and "optional" parameters must be passed as empty variants. This makes MSVC a somewhat cumbersome container for ACOP, but it can nonetheless be used, as is used in several applications in HERA control.

5 CONCLUSIONS

ACOP has an enthusiastic following at DESY as concerns both its data acquisition and its data rendition capabilities. Furthermore, it has been seen to integrate seamlessly into a number of popular container applications such as Visual Basic, Excel, HP VEE, and LabView, all in common use at DESY. As the standard desktop workstation as well as the standard HERA console are PCs running Windows NT, ACOP's current limitation to Win32 does not pose any significant development problem. As ActiveX becomes available on non-Windows systems, it is hoped that ACOP will respond with cross-platform implementations as well. In the mean time, most control physicists and engineers are now able to trivially access the control system of choice (independent of protocol), and do so from their favorite development environment.

REFERENCES

[1] I.Deloose, P.Duval, H.Wu, "The Use of ACOP Tools in Writing Control System Software", Proceedings ICALEPCS'97, 1997.
[2] Esposito, D., "Writing COM Objects with Scripting Languages", Microsoft Developer Network News, Vol. 7, No. 6, 1998.
[3] P.Duval, "TINE: An Integrated Control System for HERA", These proceedings.
[4] Ruediger Schmitz, "A Control System for the DESY Accelerator Chains", Proceedings PCaPAC'99, 1999.
[5] see: J.Hill, "Experience with PC Based EPICS IO Controllers", Proceedings PCaPAC'99, 1999 for references.
[6] http://wwwps2.cern.ch/acop,
http://www.desy.de/hera/controls/acop