The Use of ACOP Tools in Writing Control System Software

Ivan Deloose
CERN, CH-1211 Geneva 23, Switzerland
Philip Duval and Honggong Wu
DESY MKI, 22607 Hamburg, Germany

Abstract

Several institutes are making increasing use of PCs in accelerator controls. In particular, Windows NT and/or Windows 95 is already, or is becoming, a supported platform at the client-end in a variety of control systems. Notably, control systems at CERN/ISOLDE, DESY/HERA, KEK/PF-LINAC, Daresbury, ISA (Denmark), MSI (Sweden), and ESRF currently make use of Windows NT as a control system client. As all of these control systems are either object-oriented or object-based, their is a considerable overlap in their functionality and required features. This point was realized at the PCaPAC '96 workshop, and gave rise to the ACOP work group, which stands for Accelerator Component Oriented Programming. The first fruit born of this group is the ACOP.OCX (OLE Control eXtension) ActiveX control. Microsoft™ ActiveX controls are the updated version of the former OLE (Object Linking and Embedding) control specification. The ACOP control has been designed in order to support the common functionality requirements of object-oriented and object-based control systems. The binary form is shareable on all Win32 platforms.

This report gives an overview of the functionality of ACOP.OCX and its implementation at CERN and DESY, together with the progress on a compatible Java bean.

1 Introduction

At the recent PCaPAC '96 workshop in Hamburg and CAT '96 workshop in Tsukuba, it was evident that a large number of institutes are using PCs running either Windows NT or Windows 95 at the control system console level.

ACOP is a client-side component, developed as a joint project between CERN and DESY. At both laboratories it will be integrated in all current and future Win32 (Windows NT, Windows 95, 98) based control systems, beginning with the 1998 run period.

The ACOP ActiveX control is based on the Microsoft™ COM (Component Oriented Model) foundation, which implies both binary reusability and dynamic linking with its container. In turn, this implies that integrating upgrades of ACOP.OCX do not require re-compilation or re-linking. As a COM object it offers a language-independent interface to its container. The most popular ActiveX control containers are typically Visual Basic, Visual C++, Visual FoxPro, Visual J++ and the Internet Explorer Java Basic. Some other Win32 applications, such as LabView, can also act as ActiveX containers, but with more limited support.

Primarily, ACOP.OCX is a client-side control 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.

In order to be applicable to a wide variety of object-based control systems, ACOP does not require adherence to any particular naming convention or data transport mechanism.

ACOP only passes parameters and data from the control component to a required, ACOP-compliant Dynamic Link Library (DLL), called ACOP.DLL. This DLL interfaces the ACOP control to the institute specific device access calls, which are mostly provided in a home-made DLL. This strategy enables developers porting to Win32, to concentrate specifically on data exchange and not to worry about the peculiarities of OLE.

With these features, ACOP targets other institutes, which make use of object-oriented or object-based Win32 control systems.

2 The Device Access Interface

2.1 Properties

Most object-based control systems offer an object-view of the front-end hardware, where the device has a unique name and exposes its hardware-specific actions through properties and methods. The latter typically involves changing hardware settings or data acquisition. It was decided to keep the device access interface as simple as possible, respecting the individual needs. This resulted into the exposure of only two device identification properties:

DeviceName and **DeviceProperty**

Both of these properties are variable-length strings so that a wide-variety of naming hierarchies and properties can be supported. Naming hierarchies might begin at the accelerator domain level and continue through device

family to the specific device. Likewise the device property is frequently a simple string such as “POSITION” but could also conceivably be more complex for other systems. ACOP assumes that these two properties are sufficient to identify any device parameter. The issue of name resolution is not addressed at the interface level, and assumed to exist at lower level.

Similarly, ACOP exposes three device data access properties (variable length strings) :

AccessMode, AccessRate, and **AccessProtocol**

Typically, control systems employ a variety of data **AccessMode**s including synchronous or asynchronous READ and WRITE, POLLING, SMART-POLLING, etc. The control system specific identifiers are passed through this property. As a string, **AccessRate** might contain for instance a polling rate in milliseconds or simply “FAST”, ”SLOW” or even in the case of CERN/PS a cycle tag. The **AccessProtocol,** however, can be considered as a control system identifier. For instance, at DESY, it might be desirable to interface both DOOCS[1] and TINE[2] protocols from the same application. In this case, the **AccessProtocol** will indicate the compliant ACOP.DLL which specific homemade layer (DLL) to branch.

In order to allow a user-friendly way of property setting at design time, the ACOP control provides several property pages. The following picture (Fig. 1) illustrates the ‘Device’ specific page.

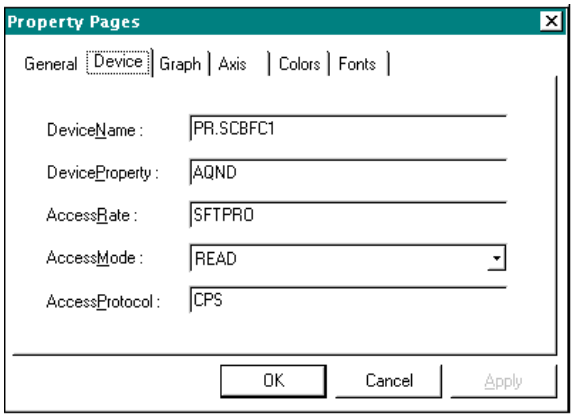

Fig. 1 : The ‘Device’ property page.

It is clear that all properties can be modified at any time during program execution.

Other properties (mostly read-only), returning information about the current device access will not be discussed here, but are documented in the software package.

2.2 Methods

We have so far only described the device access properties. To acquire data or send commands to the front end, the user must call one of the ACOP device access methods. These are listed below:

Execute

Issues a synchronous request based on the current data access and device properties, and returns a completion code. By default, the method takes the data buffer as only parameter. Optional (also called Extended) parameters are foreseen when the user wants to perform a bi-directional call. e.g. Sending some settings down to a device while doing a data acquisition.

OpenLink

Issues an asynchronous request based on the current data access and device properties, and returns a handle which identifies the open link. The user only needs to specify the array size and the data format for asynchronous data acquisition. The same optional parameters are available as in the **Execute** method in order to allow bi-directional device access.

When new data arrives, an event notification callback is triggered (see 2.3). The acquired data can now be retrieved by calling the **GetData** method.

There are no limits on the number of simultaneous open links issued by the same instance of an ACOP control.

AttachLink

Similar to the **OpenLink** method, but the data buffer is directly attached to the established link. This means that buffer is automatically updated when the notification callback gets fired. The **GetData** method is not required in this case. **AttachLink** can only be used if the memory allocated by the data buffer remains stationary during program execution.

GetData

This method should be called in response to the event notification callback routine. It retrieves the incoming data into the local buffer for links created by the **OpenLink** method.

CloseClink

This method terminates the specified link, or all active links if the argument is omitted.

In addition to all the standard OLE data types, the methods accept any kind of user defined data type or structure. In the last case, some optional parameters are

required to indicate the array size and format of the data. It is clear these user-defined types are control system specific and has to be recognized by the underlying software layer (homemade DLL).

2.3 Events

Callback in OLE controls takes the form of events. Regarding device access, ACOP exposes only one event :

Receive

This is triggered upon receipt of incoming data or interim status messages (such as link time outs).

2.4. Device access sample

The following Visual Basic code illustrates the simplicity of device access (which is especially simple if the data sets involved use OLE types). As an example, consider obtaining an array of 1200 points, representing the shape of the magnetic field of one of the CERN/PS accelerator cycles. Provided that the device access properties are filled in correctly, one writes simply:

```
Dim PSField(1200) As Single      'as global
acop.Execute PSField             'synchronous execution.
acop.AttachLink PSField          'asynchronous execution.
```

2.5. ACOP.DLL

As already mentioned in the introduction, a compliant DLL called ACOP.DLL connects the ACOP.OCX to the individual communications layer of the control system in question. Most of the data exchange between the OCX and the DLL occurs via a unique function, called DevRequest. When one downloads the ACOP package from a web-site, a sample ACOP.DLL is included, which simulates data access. Furthermore, the source code contains key wizard-like "TODO" comments to indicate where the user should fill in his private device access calls. This task is particularly simple when the concerned control system is already accessible through a Win32 DLL.

3 The Graphical Interface

One could be perfectly content to leave the ACOP control invisible and simply make use of the bevy of graphical widgets already available as ActiveX controls. Nevertheless, there are several graphical rendition styles that are used repeatedly in console applications, such as displaying traces and histograms, but sometimes, none of the graphical widgets in the arsenal available to the programmer does precisely what is required.

In order to offer the programmer more of the specific tools he needs, a control system oriented graphical interface has been included in the first release of the ACOP control. The programmer can choose from a variety of display modes, ranging from simple vector drawing to adjusted histograms. The histogram display mode itself offers several features such as displaying a marker or tagging individual lines. ACOP also features a very intuitive zooming capability. The drawing method parameters make use of the standard OLE data types, so that a method such as

acop.draw PSField,

called inside the **acop.Receive** event procedure of the device access sample from paragraph 2.4, produces the following output (Fig. 2) :

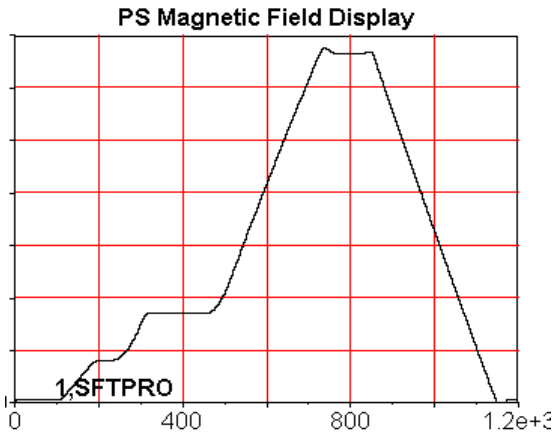

Fig. 2 : The PS magnetic field for a particular cycle

4 Conclusions

As a client-side control component, ACOP's usefulness is primarily at the device interface level. It hides the bulk of details of client-server data exchange, and separates the OLE-specific and COM-specific interface the applications programmer sees from the control-system specific data transport mechanism. It can be used as a unification layer on the client-side for a wide variety of PC based control systems. In addition a powerful graphics package is part of the control.

ACOP has been tested at both DESY and CERN and has been seen to offer a very simple and intuitive interface to control systems application programmers. At CERN, it will replace the current ISOLDE and PS device access control (RpcOle32.OCX) [3] and be deployed as standard device interface for all future Win32 based applications (e.g. The ISOLDE/REX [4] project). At DESY, it will replace Eqp.OCX, currently used in HERA and PETRA Win32 console applications.

As an object, it offers a type of inheritance called aggregation and can be sub-classed in other OLE controls. As a COM object, its interface is language (container)

independent. It can be fully integrated into a number of commercial products, including Visual Basic, Visual C++, Visual J++, Visual FoxPro and Internet Explorer. Using ACOP in the Visual Basic development environment provides the user the ability to create on-line prototypes and operational programs in a minimum of time. At the same time, it can be used in more complex environments such as C++ or Java.

Work is currently underway on an ACOP Java Bean that would offer similar capabilities. As a Java Bean, however, it will be inherently slower than a native compiled ActiveX control. As more and more institutes are embracing Windows NT as a console, the most practical control component is more likely such an ActiveX control.

References

[1] G.Grygiel, O.Hensler, K.Rehlich, "DOOCS: A Distributed Object-Oriented Control System on PCs and Workstations," PCaPAC conference 1996.

[2] P. Duval, "DESY Activities '96: A Report on PCaPAC and Status of the HERA Control System," IWCSMSA workshop 1996. Note: "TINE" was described as "MCS-1" in this report.

[3] I. Deloose, "Integrating the New Generation of ISOLDE Controls into a Multi Platform Environment", Proceedings of PCaPAC'96, Hamburg, Germany, Oct. 7-9, 1996

[4] I. Deloose, "Windows NT as Device Server for the ISOLDE-REX project", CERN internal note PS/CO Note 97-27

The ACOP package can be obtained from :

- http://wwwps2.cern.ch/acop
- http://www.desy.de/hera/controls/acop

Email: Ivan Deloose: Ivan.Deloose@cern.ch
Phil Duval : Duval@desy.de
Honggong Wu : Wu@desy.de

Filename:	P100.DOC
Directory:	D:\NPAPER97\ZRY
Template:	C:\WINWORD\TEMPLATE\NORMAL.DOT
Title:	The Use of ACOP Tools in Writing Control System Software
Subject:	
Author:	JIANG MINGBAO
Keywords:	
Comments:	
Creation Date:	11/14/97 6:51 PM
Revision Number:	3
Last Saved On:	11/14/97 6:51 PM
Last Saved By:	JIANG MINGBAO
Total Editing Time:	2 Minutes
Last Printed On:	01/19/98 4:56 PM
As of Last Complete Printing	
Number of Pages:	4
Number of Words:	1,976 (approx.)
Number of Characters:	11,266 (approx.)