ACOP AS A JAVA BEAN

Philip Duval and Honggong Wu, DESY MST, Hamburg, Germany

Abstract

The ACOP[1] (Accelerator Component Oriented Programming) ActiveX control has been extensively used in control applications not only at the DESY accelerators but at many other laboratories as well. A new ACOP Java Bean reusable component has recently been developed and tested. This new component keeps same user interfaces as the ActiveX control. As in the latter case, it offers plugs for different data exchange protocols, such as TINE[2] and Channel Access. Its graphic package was also adopted from ActiveX control, offering the same features fine-tuned for accelerator application development. The ACOP Java Bean can be used in a very similar fashion in a Visual Java environment just as the case for ActiveX in Visual Basic. Detailed development issues as well as a common user interface will be presented. In particular, we shall present benchmark results concerning ActiveX and Java Bean performance as well as ease of development.

1 INTRODUCTION

Since its release in 1997, the ACOP ActiveX component [1] has been shown to be a very useful client side control tool for writing application programs using WIN32 systems. The last upgraded version 2, released in 2001, included numerous new features, both for device access and data rendering. Given the popularity of Java natural follow-up to the ActiveX control would be an ACOP Java bean, offering the same capabilities and functionality. The initial release indeed provides the same device rendition and access capabilities as in the ActiveX control, and offers plugs for data transport, also in a similar manner. It has been tested using the TINE Access protocol for device access, and can in any case use a "Simulator" plug. Data transport "plugs" are incorporated via an interface plug class. In this paper we will emphasize the differences between implementing the ACOP Java bean and the ACOP ActiveX component, as well as several new futures (already included in the ACOP ActiveX release 2).

2 ACOP BEAN

2.1 Parameter passing in Methods

In the ACOP ActiveX control data and parameters are passed as OLE Variants, where the infrequently used parameters are all optional. Also note that as user-defined data types are not OLE type an additional data type parameter is needed for such cases. In the Java bean, however, data objects are simply passed as objects, obviating the need for a "data type" parameter. The passing of infrequently used "optional" parameters is then realized via method over-loading.

2.2 Device Access Interface

The principal data access methods, Execute(), OpenLink(), AttachLink(), GetData(), CloseLink(), GetLastDataSize(), SetReceiveQueueDepth(), are maintained with similar functionality as in the ACOP ActiveX. Differences in parameter passing follow the discussion in section 2.1. ACOP data access properties such as Status and Timestamp are likewise provided, with one difference being that the Timestamp property in the ACOP bean is a Java Date Object in the case of the bean. In addition, information querying as to available device properties from the underlying system are also provided. The latter refer to improvements found in Release 2.0 of the ACOP ActiveX control, not mentioned in [1], allowing hierarchical querying. These include:

- DeviceContext. String property used to define the accelerator facilities.
- DeviceGroup. String property typically used to define Device Servers.
- DeviceName. String property typically used to identify specific device instances.
- DeviceProperty. String property targeting a specific functionality of a device server.

2.3 Device Rendition Interface

The ACOP Java Bean provides the same sort of charting functionality and features as the ActiveX control, and with the same preference for displaying data in a control system context (such as plotting data versus timestamp, tagging of histogram elements, accepting device labels directly on the chart label, to name a few). The helper functions such as ReferenceFunction(), WeightFunction(), FFT(), GetDrawnData(), etc. are also fully supported in the bean. As a good many of the current ACOP features and functionality was introduced in Release 2.0 of the ActiveX control, we will enumerate some of them below:

- Error window. Sets a window size where out-of-bounds data will be plotted with an ERROR color.
- ReferenceFunction. Stores a reference function whereby subsequent draw method calls will show the difference of the drawing function and the reference function. Trivial shifting of the entire drawn array can as before be achieved through the YShift property.
- WeightFunction. Stores a weight function whereby drawn arrays are weighted (multiplied) by those values in the weight function. Trivial scaling of the entire drawn array can as before be achieved through the YScale property.
- Cursor Marker. Can be displayed, locked or moved with the plot.
- Fitting functions. Several commonly used fitting functions are provided, such as straight line, exponential.
- RefereshScreen. Used for updating a fraction of the plot.
- AppendScreen. Adding new data to the plot, especially for plotting newly retrieved data with timestamps.
- Leading Edge Marker. Indicates the position of appended data points.
- AcopConfig. A string property used to save and restore user predefined settings, which is similar as template setting.

2.4 Event firing

Similar to the case of ACOP ActiveX, two types of events are implemented, mouse and data events. The parameter passing, during event firing is different for ActiveX and Java due to the inherent differences in event passing in ActiveX and Java. In the latter case, all relevant data are captured inside the passed instance parameter of the ACOP event class, and thus in the AcopEventListner() the application should use getMethods() to retrieve the event data.

3 PROTOCOL PLUGS

In the case of the ActiveX control, the ACOP-compliant Dynamic Link Library (DLL), is used to interface the ACOP ActiveX control to the implementation specific device access calls. The introduction of data transport plugs in the case of the Java bean must of course take on a different form, as DLLs are not a feature of Java. In the Java environment, an abstract transport class replaces the DLL, to serve different data exchange protocols.

The initial release of the ACOP bean includes a simulation class for producing random data and which serves as an example as to what must be done to incorporate implementation-specific plugs into ACOP. The plug for the TINE transport class is also provided as a further example for data transport for an existing protocol. Of course the Abstract transport class should always be implemented to overwrite the required data transport methods.

The case of user-defined data types is however not completely handled by simply passing data parameters as Java objects. This is due to the data access and display methods which, when passed a non-primitive data type, require a way of determining the location and types of the elements contained inside the user defined type. Here one should implement the getUserData() method, which is also included in the Abstract transport class.

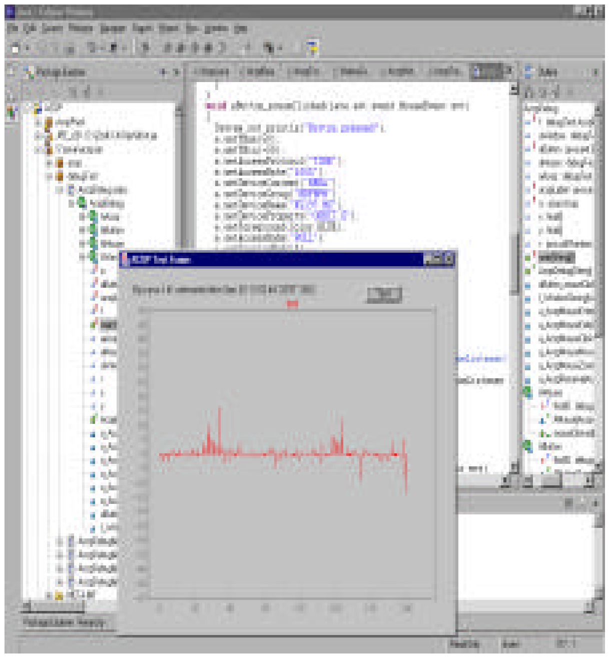

Fig 1. A simple running test application, with ACOP Java bean, plotting HERA proton orbit of one monitor using Eclipse Java developing environment

CONCLUSION

As a point of comparison we have made several simple benchmark tests regarding the ActiveX control and the Java bean. One would expect the Java bean to be slower than the native compiled ActiveX control, and it is. Rapid display of 1024 double values was possible at 100 Hz in the ActiveX control, whereas 60 Hz was achieved with the Java bean. This is most probably due to the garbage collection mechanism, taking place at inopportune times. In both cases, ActiveX and Java bean, there was no appreciable overhead regarding data access.

The ACOP Java bean is written with native JDK 1.4, and can be used in any Java developing environment.

In the Visual Java environment the ACOP Java bean is used as a simple control, one can drop it onto a frame or panel; visually manipulate it with its property page at design time. As a bean it can also be sub-classed in other Java controls. This is illustrated in the example below.

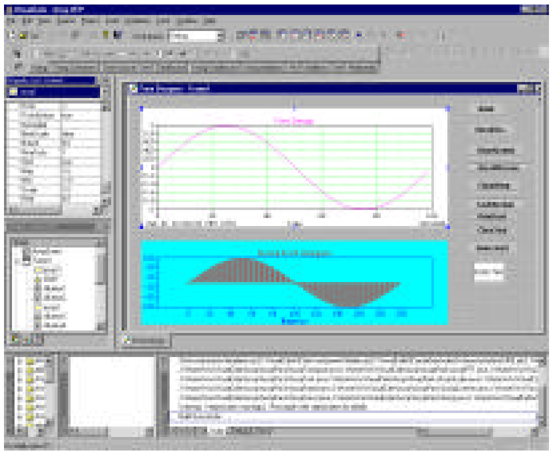

Fig. 2 Visual Café design environment using ACOP Java bean

The current release of the ACOP Java bean (at the time of this writing, a beta version) can be found at the ACOP web site: http://desyntwww.desy.de/acop.

REFERENCES

[1] I. Deloose, P. Duval, H. Wu, “The Use of ACOP Tools in Writing Control System Software”, Proceeding ICALEPS’97, 1997.

[2] Philip Duval, The TINE Control System Protocol: Status Report, Proceedings PCaPAC 2000, 2000.