# THE ACOP FAMILY OF BEANS: A FRAMEWORK INDEPENDENT APPROACH

J. Bobnar, I. Kriznar, Cosylab, Slovenia
P. K. Bartkiewicz, P. Duval, Honggong Wu, DESY, Hamburg

*Abstract*

The current ACOP (Advanced Component Oriented Programming)* controls set has now been expanded to include a wide variety of graphical java beans, which simultaneously act as displayers of control system data. Besides the original ACOP Chart, the set of ACOP beans also includes a Label, Slider, Table, Gauge, Wheel, and image control, along with an invisible Transport bean, which is itself embedded in the ACOP GUI beans. The new ACOP beans all offer design-time browsing of the control system to expedite data end-point selection. Optionally a developer can choose to connect and render the incoming data automatically, obviating the need for writing code. The developer can either forgo this option or choose to override the generated code with his own, allowing for rich client development. At the same time a user can browse and add or change the control system endpoints at run-time. If the application is using the Component Object Manager (COMA)** then all visual aspects of the application can be edited at run-time, allowing for simple client development. This scenario is independent of a framework, and the developer is free to choose the IDE of choice.

## INTRODUCTION

Console applications for PETRA III will make extensive use of Java as a development tool. In order to satisfy the demands both for writing rich-client control applications and configurable simple clients (without coding) we have extended the capabilities of the current ACOP chart bean, and are now providing ACOP functionality to a wider set of displayer beans. The new ACOP family of beans consists of a transport bean, which is responsible for data acquisition, and in our case uses primarily the transport plug for the TINE [3] protocol, and also consists of several graphic beans for displaying data. The transport bean provides device specific and graphic independent meta data, which can be browsed by the ACOP graphic beans, provided they reference the transport bean. The ACOP graphic beans themselves support popup menus (customizers) for displaying device specific transport meta-data as well as displayer-specific display properties. Drag-and-drop (DnD) is supported for passing transport and displayer meta-properties at both design time and run time. Any changes in transport or display settings introduced at run-time can then be saved and reapplied (if desired) upon the next start of an ACOP application. The current ACOP transport bean offers plugs only for the TINE protocol or for transport simulation.

## ACOP TRANSPORT BEAN

The ACOP transport API is primarily a “narrow” interface dealing with data “links” as opposed to properties with “getters” and “setters”. This proves to be a more general interface when dealing with a client API such as TINE, which allows method calls. Data Links can either be synchronous or asynchronous as discussed already in [2]. The transport API allows a three-tier hierarchy for specifying a device location, namely “Context”, “Group”, and “Name”, and a “Property” for accessing a device property or method. These entries generally specify the target endpoint of the displayer. The ACOP transport customizer will access the control system’s naming services to allow the user to browse his way to a desired endpoint. In addition, once an endpoint has been selected, information as to a device’s property-specify data is also provided. This information includes the appropriate data format, size, array type, access, engineering units and so on. A set of APIs are defined for retrieving this generic data for use in the ACOP graphic beans. The ACOP transport customizer allows setting of all such transport parameters.

## ACOP GRAPHICAL BEANS

The ACOP family of graphical displayers consists of a two dimensional chart, table, label, slider, gauger, wheelswitch and image container. They are extended from their counter parts in the Swing graphical widget set. AcopChart and Table are used as multiple displayers, which mean that they can present values of multiple device properties at the same time, while all the other widgets can show only a single value. An invisible middle layer is used for communication between displayer and ACOP transport and by setting connection endpoints directly on the displayer, it is automatically connected to a desired device property.

### *Property Customizer*

Each ACOP graphic bean has its own individual property customizer pertaining to its particular rendition features. The customizer can be accessed both at design-time and at run-time for setting display properties and the device connection properties.

The property customizer is extremely useful for browsing the available control system endpoints as well as the available display properties. If the application developer is writing a rich client, he has full control over

* http://acop.desy.de
** "The Run-Time Customization of Java Rich Clients with the COMA Class," P. Bartkiewicz, et al., these proceedings

all ACOP events and any data manipulation or filtering which should be done prior to display. If the application developer is writing a simple client, the customizer can be configured to attach the assigned connection endpoints directly to the displayer without writing a single line of code.

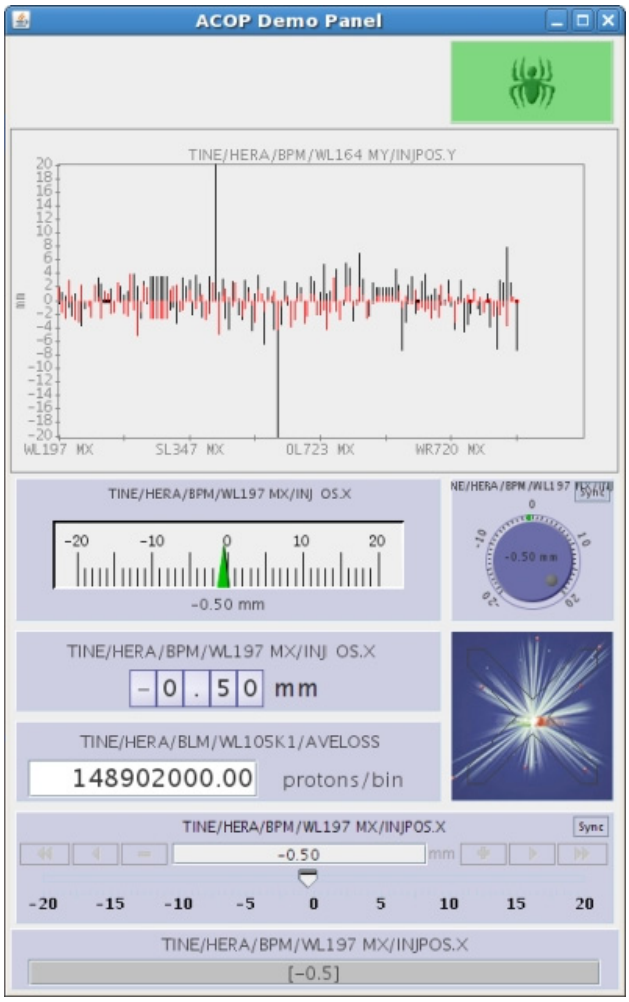


Figure 1: ACOP graphical beans.

### *Meta-Data Popup*

ACOP graphic beans all respond automatically to a 'Mouse Down' event over the display area to provide a popup display showing any and all attached control system endpoints, as well as any relevant meta-data, such as property characteristics and bean properties.

### *Drag and Drop*

All ACOP display beans support Drag-and-Drop (DnD) in the following way. A 'Start-Drag' event will collect and serialize all pertinent data transport information connected to the displayer along with the relevant display information. For instance an ACOP chart be displaying the time histories (trends) of both the beam current and lifetime. A 'Start-Drag' event will contain all the endpoint information (device context, group, name, property, etc.) for both the beam current and beam lifetime. In addition it will contain the display settings such as the display colors, max and min settings, etc. used in the chart. Another application receiving this information in a Drop event can make use of it in a context sensitive way. For instance, dropping into Notepad will just yield a text representation of the serialized data. Dropping into a History Viewer might collect the endpoint information and obtain the long-term histories of both the dropped endpoints and append them to the current history display. Drop into another ACOP chart will reflect in the second chart having the same settings as the drag source. Dragging between two different ACOP displayers can of course lead to a loss of display information. For instance, a chart is a more complex object than a label. Hence dragging from an ACOP chart to an ACOP label will preserve only the control system endpoints and the display settings that label can make use of. The above case of an ACOP bean connected to the beam current and beam lifetime will pass both endpoints to the ACOP label, however, as the label is a single displayer as opposed to the chart, which is a multiple displayer, the user will have an option to choose the endpoint which should be connected to the ACOP label. The color, line style etc. are of no use to the label, and it will therefore jettison them.

In addition, if for instance a multi-channel array is being displayed in an ACOP table, the array index at which the Drag event is initiated is also passed. An ACOP label or History Viewer application receiving the ensuing Drop event will then target the individual channel being dropped.

## ACOP SPIDER BEAN

The AcopSpider is a visual debugging tool and can be used as a part of any application. It is a graphical component, which informs user of any problems that occur within the application (eg. exception reporting). When put in a Container, the component automatically registers itself as a TINE link listener and presents the status of all currently active links. In the case of an error user is notified via the change of look of the bean. If the error is related to the link that is currently being attached to some displayer, then the AcopSpider can also provide overlay decorations on that particular displayer, which are instantly detected by the user.

## VISUAL EDITORS

All ACOP beans follow the Java Beans conventions. These conventions are not part of the Java Beans API, but in many ways they are more important than the API itself. These conventions are sometimes referred as design patterns and they specify such things as names and signatures for property accessor methods defined by the bean.

The reason for these design patterns is interoperability between beans and the beanbox programs that manipulate

them. One sort of these programs are the visual editors or GUI builders. Such builders make use of the conventions to enable user to create and modify components and graphically design the application. Using the builder supplied property sheets or the aforementioned customizers one can set any property of such bean. The customization of the bean is immediately reflected in the generated code, which makes the ACOP framework very useful when creating simple clients. One does not need to know much more than only basic Java commands and is already able to create simple control system applications. The majority of the ACOP beans properties can be later changed at run-time using the same customizers which were used in design-time.

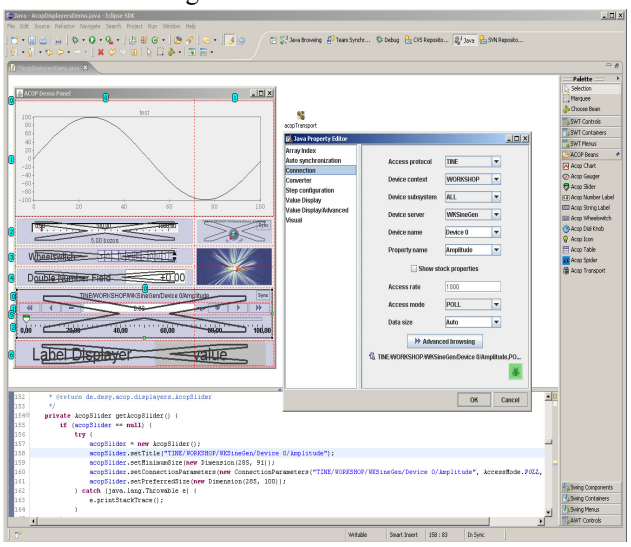

Figure 2: Using ACOP beans in Eclipse's GUI builder.

For even more efficient GUI building the ACOP library has been extended to include an additional Eclipse Visual Editor plug-in – a palette, which enables easy browsing of ACOP beans. The ACOP library also includes configurations, which enables the same browsing feature in other GUI builders such as NetBeans.

### *Using COMA*

Component Object Manager (COMA) defines another project type where the java beans conventions will be used. COMA is itself a lightweight application framework, which enables the user to create or modify applications in run time. The simplest COMA application is an empty JFrame. During run-time, one can browse a vast set of Java beans provided in the classpath of the COMA and add them to the frame as desired. Besides creating a brand new application, modification of an existing application is also allowed by this framework. Because the ACOP beans follow the Java beans convention, it is therefore very easy to create a simple application without using heavyweight GUI builders or even opening an IDE for that matter. All modifications made to a COMA application can be stored into an .xml configuration file, which can then be loaded later, thus preserving all modifications. Thus COMA plus ACOP beans provide a lightweight alternative to heavyweight frameworks.

## ACOP-BASED APPLICATIONS

ACOP beans have been successfully used in number of simple as well as more complicated applications. Besides the numerous control panels which will be used by control system operators, much more demanding applications, based on ACOP library have been created, such as the TINE Archive Viewer and Multi-Channel Analyzer for example. These applications usually employ certain wrappers around the beans in order to provide the required functionality, however the library still represents the backbone of all current and future applications giving all of them the same look and feel.

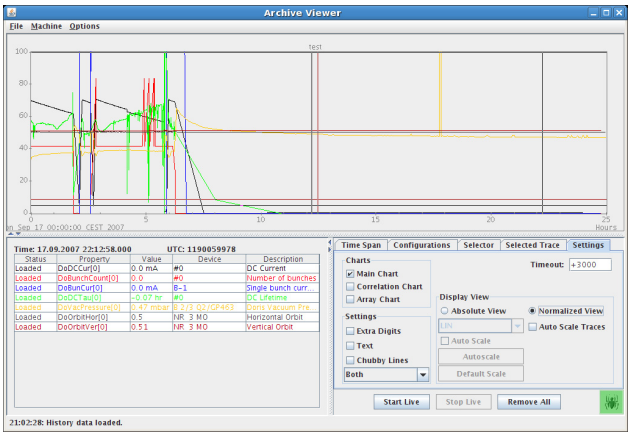


Figure 3: Archive Viewer showing DORIS overview in an ACOP Chart.

## CONCLUSION

Writing simple clients for a control system is not a very demanding task, however it can be very time consuming due to the large number of applications that are required. In this case ACOP library can help tremendously because with its help such applications can be created by just a few mouse clicks. Furthermore, (at least for the variety of control applications which do not require 'business' or display logic) one does not need to use an experienced programmer (potentially costing even more of his time explaining the application requirements). Instead the engineer, physicist or operator can easily create client applications by himself, just as he desires.

## REFERENCES


[1] I. Deloose, P. Duval, H. Wu, "The Use of ACOP Tools in Writing Control System Software", Proceeding ICALEPCS'97, 1997.

[2] Philip Duval, Honggong Wu, "Acop as a Java Bean", Proceedings PCaPAC 2002, 2002.

[3] http://tine.desy.de