ACOP.NET: NOT JUST ANOTHER GUI BUILDER

P. Duval, M. Lomperski, J. Szczesny, H. Wu, DESY, Hamburg, Germany
T. Kosuge, KEK, Tsukuba, Japan
J. Bobnar, Cosylab, Ljubljana, Slovenia

Abstract

ACOP (Advanced Component Oriented Programming) tools have been useful in control system GUI application development for some time, originally as an ActiveX component [1] offering a transport layer and a multi-faceted chart and then later as a suite of components in the Java world [2]. We now present a set of ACOP components for development in .NET. And where the emphasis in the past has been primarily on rapid application development of rich clients, this new palette of components is designed both for fully featured rich-client development in any of the .NET supported languages (C#, C++, VB, F#) as well as for fully configurable clients (with design-time browsing), where no programming on the part of the developer is necessary, and of course for any combination between these extremes. This is an important point, which will become clear when we contrast application development with ACOP.NET with other control system GUI builders such as Control System Studio (CSS) [3] and Java DOOCS Data Display (jDDD) [4]. Although Visual Studio is the GUI builder of choice, we will present other available options, for example on Linux. Examples using transport plugs for TINE [5], STARS [6] and EPICS Channel Access [7] will be given.

INTRODUCTION

The control system for a particle accelerator or other large facility must meet stringent requirements for stable operation. And it must offer diagnostic tools for spotting, finding and fixing problems as well as online and offline analysis tools for examining machine data and improving operations. This much goes without saying. Yet, the control system is often judged by its operational interface, i.e. at the presentation level.

There are several strategies for providing the operators, engineers, and machine physicists with useful control system applications. These applications can be provided by a controls group. Or application development tools can be given to the end users so that they might generate the controls applications themselves. Or … both. And note that the three groups of end users mentioned above will each have a different perspective as to what a controls application should be able to do. They are also likely to have different skill sets concerning programming abilities and understanding the various aspects of the machine being controlled.

A very common strategy is to have a controls group provide certain core applications, but to allow the operators and engineers to create control panels using some framework, where no programming skills are required, and where, more often than not, programming is not even possible. This is for instance the case of CSS, Taurus [8], and jDDD, each of which provides the panel developer with a rich set of displayer widgets, which can be attached to a control system address, and combined with other graphical widgets providing some display logic. Here it is assumed that there is no need to program any additional display logic, although this would be nominally possible with Taurus in a Qt environment.

Machine physicists, on the other hand, frequently need higher level control in order to 'test things' and improve the overall performance of the machine. That is, they need to be able to program at the client side. A common strategy here is to provide control system components in a mathematically based programming environment such as MatLab or Python (and NumPy).

This two-pronged approach, a panel builder for basic display and control applications, and MatLab or Python support for high level controls, is common to many accelerator facilities.

Yet another strategy is to support rich client development using rapid application development (RAD) tools. This approach has met with great success at HERA and PETRA-3 where RAD tools in either Visual Basic and ACOP ActiveX or Java and ACOP beans were used by both the controls group and machine physicists to develop applications [9].

We shall now describe ACOP.NET which provides a single application development paradigm, offering both a non-programming panel building environment as well as a fully-programmable environment (and of course any combination between these two extremes). In fact, a panel application without a single line of user code can be extended by supplying additional logic at any later date.

ACOP AND .NET

ACOP was originally designed as a RAD tool in the ActiveX world [1] and was predominately used in Visual Basic applications. It featured a very powerful chart, with a number of control system oriented features and a transport layer. It was later ported to java and expanded to include a variety of displayer beans suitable for rich client development in java [2].

Now, in general, rich client development in java requires more extensive programming skills than programming in Visual Basic. Thus, it is very tempting to offer a panel building framework with smart widgets which can be configured at design time and remove the programing aspect entirely from the application developer. This is in fact the approach of both CSS and jDDD, where an application might exist as an XML file

which is interpreted and rendered at runtime. Taurus likewise makes use of an XML configuration file which is rendered at runtime, but here the platform is Python rather than java.

The latest variant of ACOP, which we now present, also offers a panel building paradigm with smart widgets configured at design time. The developer need not write a single line of code, but does need to choose a .NET programing language. The finished ACOP application will exist as a single executable file. The development project consists of a .NET *solution* and code modules (maintained by the Visual Studio designer) instead of XML files. The executable is every bit as portable as a java application and can be run under mono [10] on non-windows systems.

As noted earlier, not all control system application needs can be met by pure configuration (no matter how many '*CALC*' records or properties are invented on the server side). A team responsible for high level controls will typically require an interface to Python or MatLab in order to be able to 1) program and 2) to make use of the available mathematical packages.

And here the beauty of ACOP.NET comes to shine. Any ACOP.NET application can instantly become a rich client (and easily leverage third party mathematics and other packages). The developer will be able to choose among such languages as Visual Basic (VB), C#, C++/CLI, and F#, tailoring an application to his own programing preferences. Thus, some thought should go into the question as to which language to choose for a particular controls project, even if it might be destined to remain a simple configured panel. And, as an aside to Python aficionados, note that Python and .NET can be combined via Python for .NET [11].

.NET, Visual Studio, and Mono

Microsoft .NET enjoys a wide acceptance and client base in the industrial and business world, if not in the control system community. Applications written in .NET on a Windows platform can generally be run on a Linux or Mac platform as is via *mono* (which is now technically sponsored by Microsoft). Although ACOP.NET GUI applications can in principle be written using mono developer on a Linux platform, it is strongly recommended to make use of Visual Studio, which features an integrated designer capable of writing and isolating the GUI component relevant code. As the *community* version of Visual Studio can be downloaded and installed free of charge this should not present any cost or licensing burdens on the developer.

The current version of ACOP.NET makes use of WinForms, which are mature, wide-spread, and encompass an extensive variety of common components.

Real vs. Graphical Programming

As is the case with any panel builder the developer can configure a useful, working application by simply setting design time properties in the GUI builder (here Visual Studio). ACOP.NET offers design-time browsing of the control system, so that the developer is not required to know a priori the end-point addresses he wishes to attach to the ACOP.NET displayers.

Often, an application can only go *so far* by merely attaching addresses to displayers before some simple display logic is desired. This can be something as mundane as changing display characteristics based on a condition, or applying some trivial manipulation of the associated data before it is displayed.

If there is no ability to actually code something, then the GUI builder might solve logic problems by offering additional logical widgets (e.g. jDDD), which is tantamount to offering a form of graphical programming. Problems of data manipulation (because the data acquired from a control system address is not in the form needed for display) are then passed on to the front end, where new properties (or CALC records) are invented to make the displayer's life easier (although not necessarily the front-end developer's).

None of this is necessary with ACOP.NET, as simple logical decisions can be coded simply in the developer's favorite language, just as minor data manipulation prior to display can be coded easily. And when more extensive experimentation with front end data is required there is effectively no alternative to programming.

The ACOP.NET control widgets can be used as is in any application and these include shape widgets which might be useful in synoptic displays. These can be used side by side with the standard .NET or other 3rd-party components.

Runtime Features minus Popup Pollution

The ACOP controls do not themselves launch independent, non-modal windows. Thus there is no danger of inadvertently filling the desktop with disjointed display windows (Popup Pollution). Launching a separate daughter window, if that is in fact desired, requires a tiny bit of code, where the click event of the launching component is used to show the additional ACOP Form.

On the other hand many controls such as the chart do offer context menus whereby various properties can be edited, or settings applied at runtime.

Should an ACOP control be bound to a control system address which is not responsive, an initial modal window is displayed alerting the user to this fact. Likewise, some components will alert the user to a failure via a modal window, for instance, when an access lock cannot be obtained, or a setting cannot be applied.

Data acquisition errors are otherwise displayed, where possible, within the ACOP displayer component in the given *error color* and *blinking* state. In addition for many components a specific *error value* can be supplied, which will supersede the last displayed value upon any data acquisition error.

Some ACOP components are in fact designed to interact with other ACOP components. This is particularly true of the ACOP Chooser, which allows a user to make an end-point address change (such as

changing the targeted device or context) at run-time and apply the change to those ACOP components designated at design time. The ACOP Status Bar can likewise set the enabled state of a list of designated components from *true* (expert mode) to *false* (read-only mode). Finally those ACOP components which can apply settings can also set selected properties of specific components in lieu of control system end points. For example an ACOP Wheel Switch could be used to set the Y-Axis Maximum of a chart.

If no explicit tool tip text is provided for an ACOP Component, it will offer the targeted end point address. This is not true of the ACOP Chart however (except for the chart frame). The application designer can instead specify an auto-link-update mouse move tool tip where the displayed data vs. horizontal coordinates or simply the chart coordinates can be displayed.

ACOP Components

As noted earlier the preferred framework for building ACOP applications is Microsoft Visual Studio. When making use of one of the ACOP project templates, the ACOP components should already be included in the toolbox, as shown below in Fig. 1. New Acop Forms can be added via the project's *Add New Item* context menu.

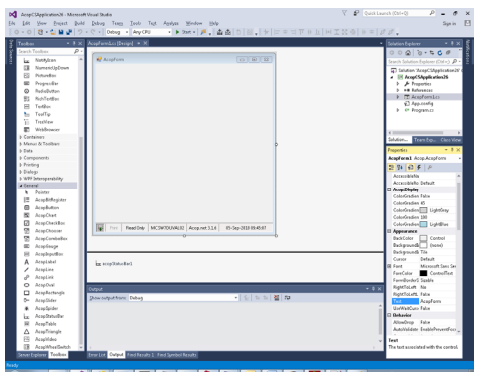

Figure 1: Starting a new project: an empty ACOP application.

As can be seen in the figure above, the basic ACOP components consist of 20 GUI components plus a non-GUI ACOP Link component which can be used in situations where explicit programming is necessary. Of these 20 components, there are two specialty components, the ACOP Spider and the ACOP Status Bar. The ACOP Spider is useful as a debugging portal, where specific link information is made available. And the ACOP Spider is itself contained within the ACOP Status Bar, which serves as a menu footer when placed on an ACOP Form (and is included by default on the primary ACOP Form when making use of an application template).

In addition, there are several shape components, which can be used in synoptic display. See Fig. 2 below. Synoptic displays often make use of clusters of components which are repeated on a panel's façade. The application designer can make his life easier by either selecting the entire cluster and copy-and-paste or by including the cluster in his own .NET component (Project Solution category = *Class Library*), which can then be used and re-used in any ACOP application.

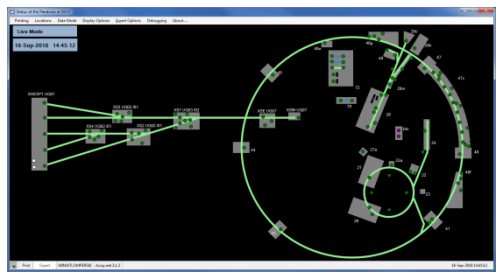

Figure 2. A synoptic display example.

Control System Plugins

ACOP components offer a control-system independent transport interface to obtain and display data from the attached control system end points. A supported control system protocol needs to provide a .NET plugin library which implements the ACOP Transport interface. Over the years there have been many, many transport layers which purport to provide a common interface to various known control systems, and we will not name them here. In addition to this requisite feature, an ACOP transport plugin should also provide browsing instructions and transport information and statistics, useful in debugging an application.

Although some ACOP components might make use of the browsing interface at runtime (such as the ACOP Chooser), this feature is mostly called upon at design time, obviating the need for the developer to be aware of and key in control system end-point addresses.

The transport information interface is used to provide a snapshot of the application's connectivity via the ACOP Spider, which is embedded in the ACOP Status Bar but can also be placed independently on any ACOP form. A green spider indicates no connectivity issues and a mouse click on the spider will launch one of the few non-modal windows automatically available to an ACOP application, where the application's connection tables can be examined in more detail. An example of connection information window is shown in Fig. 3 below.

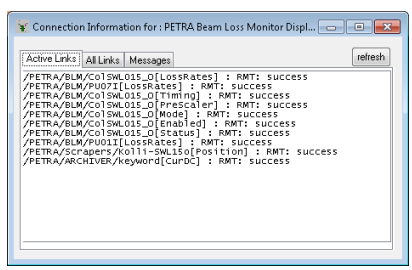

Figure 3: An example of the connection information made available by the ACOP Spider.

The end point address paradigm used in the ACOP transport API is one of a 4-tiered hierarchy, following that used by the TINE control system as well as DOOCS [12] and TANGO [13]. The top level is the Link Context,

followed by the Link Server, the Link Device (which together form the link end-point address), and the Link Property (which specifies the method or attribute to be accessed at the end point). As this does not directly fit some control system protocols, an ACOP plugin can turn off browsing and make use a simple Link Address or provide a mapping into the ACOP hierarchy. For example, Channel Access [7] does not provide any systematic hierarchy other than a flat namespace appended by a known set of meta-property decorations. STARS has a more-or-less open hierarchy. In such cases one can simply supply a Link Address without browsing or make use of extra naming services specific to the control system site (e.g. where control system addresses are maintained by LDAP or other file system services), in which case the ACOP plugin used for a particular control system protocol would be site specific.

To this end, ACOP.NET is somewhat TINE centric, as it makes allowances for features available in TINE (such as multi-channel arrays, property-oriented server browsing, structured data, etc.) which are not available in other control systems. This is hardly a short-coming and is in analogy with CSS being EPICS centric, jDDD being DOOCS centric, and TAURUS being TANGO centric.

In any event, an ACOP transport plugin must be written as a .NET DLL which utilizes the *AcopPlugin* interface. This is shown in Fig. 4.

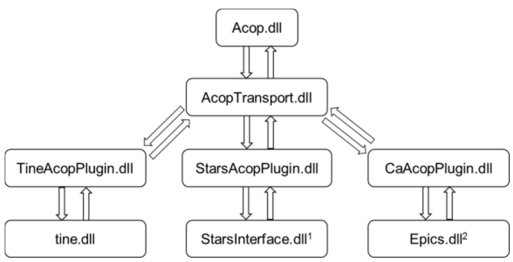

Figure 4: Acop Transport plugins.

An application can make use of multiple plugins if desired. Plugin information, such as the default protocol for an ACOP widget or design-time browsing instructions, can be supplied in the *AcopTransport.dll.config* file, deployed during setup and installation.

APPLICATION BUILDING

As mentioned earlier, the most sensible method of building an ACOP application is by making use of some flavor of Microsoft Visual Studio. In addition it is recommended to make use of one of the ACOP application templates for C#, VB, or VC++ development. The choice of language might seem irrelevant if one is not intending to program. If however, the application is likely to be long-lived, a modicum of thought concerning the programming skills of those who might want to add a finishing-touch at some later stage is nonetheless encouraged.

Installing ACOP.NET

ACOP.NET is most easily installed on a Windows machine by making use of the *Setup* utility found on the ACOP.NET Web page [14]. A pre-requisite is an existing installation of Visual Studio. Depending on the ACOP plugins selected, the corresponding control system should also be installed on the local host.

Following the installation, a new Visual Studio project will offer an AcopCSApplication in the C# Templates category and an AcopVBApplication in Visual Basic Templates. Selecting one of these will produce an empty application panel resembling that shown above in Fig. 1.

Configuring Simple Panels

Starting with the blank slate shown in Fig. 1 above, one then places the GUI components one wants on the empty panel, as with any modern GUI builder. The ACOP components have their own properties and events, some of which are specific to display and GUI interactions, as well as others pertinent to data acquisition. The latter fall into the category *Acop.Transport*, which is shown in the designer property grid in Fig. 5 below.

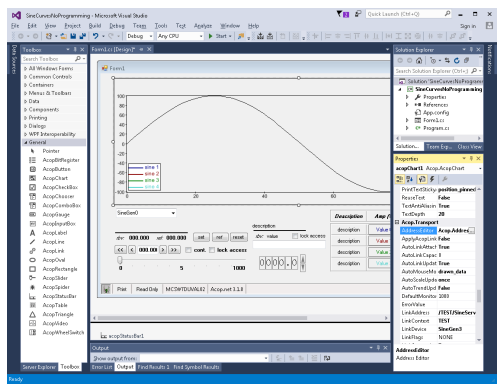

Figure 5: An ACOP application in design mode, highlighting the Acop. Transport category.

It is here that design time browsing can greatly assist in finding a control system end point address and its corresponding property display characteristics, although these values can also be keyed in by hand in the designer.

Further help for the application programmer is provided through various modal designer editors which allow multiple settings of ACOP properties at the same time. For instance the Address Editor, shown below in Fig. 6, is available to all ACOP GUI components and allows browsing and selection of multiple link properties in a logical and intuitive manner, in place of setting these properties one by one in the property grid.

Otherwise each ACOP component will offer specific display properties in the designer property grid.

If synoptic displays are desired, the ACOP shape components can be utilized for basic shape design. These are likewise smart components, which can display control system data as well as adjust display settings such as fore and back colors and the blinking state based on

comparing incoming data to given thresholds. More complicated shapes can easily be built from the basic shapes.

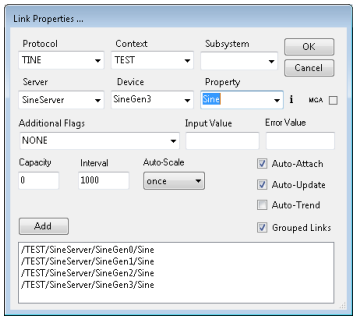

Figure 6: The design-time ACOP Address Editor.

Adding Code

One can generate a powerful and fully operative control system application via the simple panel configuration method described above. And it is easy enough to take the application to the next level by adding a smattering of code, which just might help render the application a bit more intuitive and functional to the end user.

An event delegate can be supplied for any of the common GUI component events (such as mouse click, or mouse move, etc.) as well as ACOP transport events (such as Link Update or Link Error). There are likewise numerous ACOP API calls to obtain the control system data so that the application might make use of it.

For instance if the value at the cursor position over an ACOP Chart display is needed in an ACOP Table cell, this could be achieved as follows: 1) Get the drawn data by supplying a link update delegate; 2) Catch the mouse move over the chart with a mouse move event delegate, which also provides the drawn data array index; 3) Add this element to in the desired ACOP Table cell.

Runtime Options

Various ACOP GUI properties and actions can be also be accessed at runtime, via associated context menus. And as noted earlier, the ACOP Status Bar provides the ACOP Spider and its runtime debugging capabilities. It also provides runtime printing options (including print to logbook) as well as application mode settings (read-only vs. expert).

CONCLUSIONS

There are of course numerous packages which can be used as control system GUI builders. CSS and jDDD are java-based panel builders, which do not offer the ability to program (although they are both able to launch independent applications or scripts). Taurus is python based and allows simple panel building (with some hidden python programming) as well as additional programming (in python) at an expert level through the Qt designer. In addition, PyQt, MatLab and LabView all offer GUI building with extensive programing capabilities (as long as there is an interface to the control system), but do not offer simple designer panels.

ACOP.NET on the other hand provides a GUI building package which can cover the entire spectrum from simple designer panel to complex application with as much computational programing as is desired and offers design-time browsing of the control system. When one does need to or wish to add code, this is again aided by the Visual Studio designer providing event delegates and intelli-sense information, and can be done in one of several languages suited to the developer's needs. The available languages range from Visual Basic (for those whose programing skills are not particularly pronounced) to C# (for the java aficionados) to F# (for those who prefer functional programing), along with C++/CLI for C++ purists. Additional packages such as Math.NET and Python.NET can provide the user with a rich development environment indeed.

ACKNOWLEDGEMENTS

We would like to thank Graham Cox, STFC Daresbury Laboratory, UK, for his assistance in providing the EPICS Channel Access .NET library [15] for discussions concerning its integration into ACOP.NET.

REFERENCES

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

[2] J. Bobnar, *et al.*, "The ACOP family of beans: the framework independent approach", *ICALEPCS'07,* 2007.

[3] CSS website, `http://www.csstudio.org`

[4] jDDD website, `http://jddd.desy.de`

[5] TINE website, `http://tine.desy.de`

[6] STARS website, `http://stars.kek.jp`

[7] EPICS website, `http://www.aps.anl.gov/epics`

[8] Taurus website, `https://taurus-scada.org`

[9] P. Duval and H. Wu, "Using ACOP in HERA Control Applications", *PCaPAC'00,* 2000.

[10] MONO website, `https://www.mono-project.com`

[11] Python for .NET website, `http://pythonnet.github.io`

[12] DOOCS website, `http://doocs.desy.de`

[13] TANGO website, `http://www.tango-controls.org`

[14] ACOP.NET Website, `http://acop.desy.de`

[15] G. Cox, "A .NET Interface for Channel Access", *PCaPAC'08*, 2008.