The Babylonization of Control Systems

P.Duval, Z. Kakucs – DESY; M. Kadunc, I.Kriznar, M. Plesko, A. Pucelj, G. Tkacik – JSI and Cosylab, email: mark.plesko@cosylab.com, http://www.cosylab.com

Abstract

The standardization in PC and network technology has produced a distinct (for want of a better term) "babylonization", where islands of control exist in perfect ignorance of each other even though they might belong to the very same facility. This is due in part to commercial equipment, which often comes with its own control software, and to the many excellent but different solutions for control systems, which have been developed in the accelerator control community.

A control systems integrator frequently has to make decisions with long-term and far-reaching consequences. Often a pragmatic approach is to allow resourceful engineers to use the best available tools to solve controls problems and then to integrate their solutions into the control system. It usually turns out that integration, if not done systematically, amounts for the largest part of the work. There are usually many ways to do this, for instance defining a software bus, using gateways, or simply allowing apples and oranges to peacefully coexist. In this paper, we will examine most of the available tools in our community for the integration of control systems, detailing the merits of each approach as well as some popular controls systems and components. We will demonstrate that it is possible to mix them in order to benefit from the best part of each.

1 AVAILABLE CONTROL SYSTEMS

There are several competing control system (CS) components, who look very similar but in fact address quite different issues in different ways: EPICS, COACK, TINE, DOOCS, ACS, TANGO, ACOP, CDEV, Abeans, CosyBeans, XAL, Databush, just to name those that are advertised as packages[1].

The different coverage of control system packages is shown in figure1. It cannot emphasize the features and services that are provided. We have therefore prepared a table, with input from authors and users of the respective packages. The table itself would exhaust the page length requirements of the proceedings. It is nonetheless illuminating and we therefore refer the reader to reference [1] for a full comparison and allude to certain aspects below.

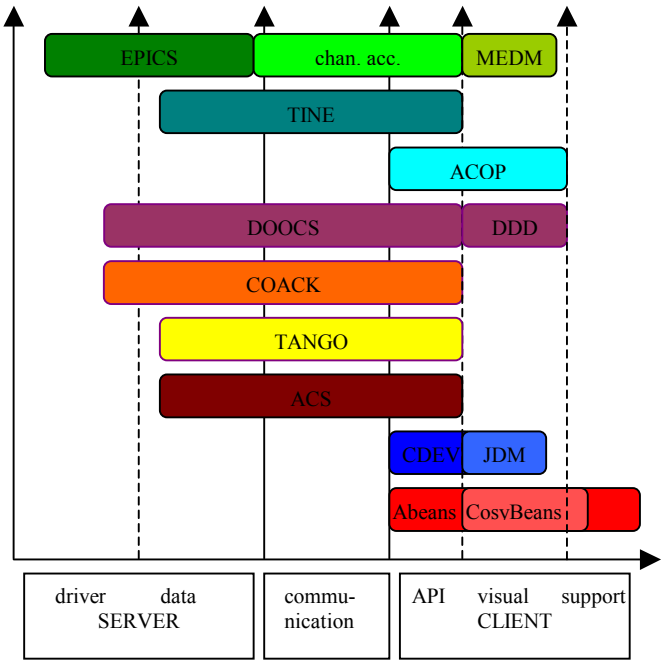

Figure 1: A comparison of control system packages and the layers they cover.

To illustrate the difficulties (and dangers) of making comparisons such as these we note that, just comparing TINE and EPICS is already like comparing apples and oranges. TINE is more of a communication protocol and should be compared to channel access. Note also that the EPICS database is really at the lowest level of the control system. One should be aware of this point, because when people say EPICS, they mean the whole lot of very unrelated things like the database, the channel access protocol and the MEDM GUI tool. The database is a viable idea and - apart from some historic glitches that are being addressed in the upcoming versions, like the short limit for names, poor debugging options - a useful approach for I/O integration. Such low-level IO integration is frequently not found in CS packages, DOOCS being a notable exception, where similar concepts are also in place. The problem for a

[1] For the sake of example we will be mentioning only some systems This choice does not represent an endorsement by the authors, nor is it any reflection on anybody else's system. As this paper concentrates on control systems, we will also not further discuss XAL and Databush, which are packages for machine physics calculations. They deserve a paper on their own.

CS integrator might be the EPICS extensions, which one is forced to use by taking EPICS or one is forced to develop with the limited API that is available.

As we see, each package has certain advantages, unmatched by any other package. So, apart from simply allowing religious freedom to reign, where each engineer can use his preferred package (but the systems coordinators nonetheless have to get the accelerator to operate), there are actually good reasons to mix the control systems in order to get best-of-breed services and applications.

2 TRANSLATORS OR INTEGRATORS?

When the control system coordinator is faced with the problem: "How do I make my apples look like oranges," he can take one of three tacks. 1) Write an 'apple-to-orange' gateway, which is a separate process utilizing the client/server APIs of both systems. 2) Use client-side 'apple-plugs' so that while client program developers think they are talking to oranges, they are really speaking native 'apple'. 3) Use server-side 'orange-plugs' so that server IOCs think they are being addressed by apples but are really speaking native orange.

Whereas each approach might have its time and place, most benefits occur for case 3 (server-side plugs).Here one knows that the server-side systematics (local alarm server, local history server, queries, etc.) are guaranteed to be there. The data in this case are as close to the source as possible.

Client-side plugs are also attractive and perhaps the next best thing. However, if the server-side systematics are not covered, you come up empty. Gateways can also solve data acquisition problems but tend to bring a host of intermediate problems with them (e.g. connectivity problems might be more difficult to locate if there is another link in the chain).

Also note, in the case of client-side plugs, if the plug you are using doesn't cover the functionality of your system, you lose! For instance, with TINE, data transfer occurs through data "links," where the access mode can be specified. Thinking in terms of "monitors", you can specify the kind of monitor: Do I want 'send on change'? (the classic EPICS monitor), or do I want 'send on poll'?, or do I want the monitor as a network subscription? (a real multicast to my multicast group), or do I want the monitor to go over a persistent TCP connection? With client APIs such as CDEV, with simple monitorOn() and monitorOff() methods, if would be difficult if not impossible to define these different categories of monitors as a developer .

In general, plugs allow you to use your preferred applications, but you are limited to existing services and tools, but this is exactly the area, where everybody has weak points. Wouldn't it be nice to use the best tools for each single application? That requires just a translator (server-side plug) to each CS package at the lowest possible level. This might present a psychological barrier for some control system coordinators, as it might at first be perceived as a potential source of instability or a security breach. Where servers are not more feature-rich than the plug, a less intrusive way of adding features would in any case be through client-side API plugs.

2.1 EPICS, TINE and DOOCS Translator

Suppose we want a TINE view of the EPICS IOCs in the system. We can

1) run EPICS2TINE directly on the IOC or
2) set aside a dedicated machine which interfaces to the IOC via channel access and runs a TINE server process for the TINE view.

The first case doesn't speak channel access at all and accesses the EPICS database directly (and is thus a translation layer on the server) and the second case is a true gateway. In a similar vein, the current DOOCS servers are bi-lingual offering the traditional SUN RPC interface as well as a TINE interface. Indeed DOOCS can run entirely on TINE (or rather TINE can run in a DOOCS context). This approach is in contrast the external gateway approach traditionally used in the past.

With EPICS2TINE, we have also elegantly solved the 16 Kbyte barrier (i.e. 4000 floats) of the old EPICS release, which has bothered us here at DESY, while using EPICS to handle certain transient-recorder archive channels (which have arrays of data which far exceed this). Thus EPICS IOCs are immediately available to say DOOCS DDD clients. Using TINE2EPICS, the DOOCS IOCs are likewise available to EPICS MEDM clients. Pure TINE clients can of course access either. Likewise, running Abeans with a TINE plug will see all IOCs as TINE servers irrespective of their parentage.

2.2 Abeans plugs for TINE[2] and EPICS

The Abeans and CosyBeans offer many advantages and features for developing client applications as described in detail in [2]. Any CS protocol and model can be attached to Abeans through their pluggable interface. Cosylab has thus developed a TINE plug for DESY and an EPICS plug for the SNS (Spallation Neutron Source at the Oak Ridge National Lab).

DESY is interested in Abeans mainly because it provides a rich framework for running TINE client

[2]At DESY Windows GUI applications make use TINE on ACOP or a native Visual Basic API In the true "babylonization" spirit, ACOP has in fact also been fitted with both TINE plugs and Channel Access plugs, but is much simpler in scope than Abeans.

applications on non-Windows machines, while keeping access to the full TINE API and services. SNS first developed XAL, which is both an API to EPICS and a machine physics package. SNS now wants Abeans as a layer between XAL and EPICS, because many of its capabilities are not presently available in XAL, nor are they being pursued due to limited manpower.

Abeans allow different models to represent the structure of the control system. Models use plugs to get data from a specific control system. At DESY and the SNS, we used the Abeans "channel" model (i.e. a narrow interface access model), which consists of namespaces and channels, to create a plug to the TINE Java class, or to the JCA EPICS class, respectively.

The following general guidelines were adopted when deciding how to resolve the integration in both cases:

1) Full encapsulation: Abeans are built as an application framework and data access layer. Abeans also define the Channel concept, both for TINE and EPICS; so no details of the TINE or EPICS layers are visible through Abeans.

2) Retain functionality: However, Abeans must provide the full CS functionality to the application programmer, not just a common subset. This has been solved with runtime plug-in services. Abeans provide at least one CS-independent default implementation, while allowing the addition of a plug-in to access a certain CS-specific remote service, such as the TINE archiver, an ORACLE database, etc.

3) Generic vs. specific: Generic solutions to given problems are preferred. Thus a configuration system that can save to any target (local or remote file system, XML or other format) is preferred to an implementation that is tied to a certain technology / approach / library.

4) Standard Java solutions: If standard Java solutions become available for problems addressed by either Abeans, TINE or XAL, these should be used, even if it means jettisoning tried code. An example is the Java logging API introduced in JDK1.4.

5) Code decoupling: TINE, EPICS and XAL functions have been introduced into Abeans. This corresponds to merging functionality, i.e. transferring to Abeans the application framework from TINE and XAL. XAL would then remain as a pure accelerator physics package. This approach produces higher quality code and there is less of it to be maintained.

2.3 A Future Scenario

The "best of all possible worlds" surely means different things to different control systems coordinators. Thus there are many examples of mixing and matching that are not only possible but make good sense.

Consider the following: We integrate VME I/O cards with EPICS (because it has the drivers), use TINE as the access protocol (for mulitcast capability), DOOCS DDD or COACK (for developing synoptic GUI panel), and ABeans/CosyBeans (for a deviceTable). One can still display the EPICS alarm table via channel access. Any number of applications using Java + ACOP, or Abeans, or MEDM or Visual Basic + ACOP could run independently and in harmony.

Further possibilities include using advanced features such as the TINE archiver, ACS logging, Abeans resource loading, etc. We should think more about the services of our systems that could be used in a generic way by other control systems.

There is not much need for competition on the system level – all CS package developers should rather work hard to get good general-purpose applications and tools. Because this is the area, where we are the weakest.

3 CONCLUSIONS

Maybe in the near future, we won't have to compete, but can choose a component that is best for a particular problem thanks to the integration tools such as TINE2EPICS or Abeans plugs. To return to the apple-and-oranges metaphor: choose your favorite, but if you have to mix apples and oranges because you have apples but someone has this great orange from which you could really benefit, then it's no big deal when there are ready solutions to make an orange look like an apple.

Staying with metaphors: there is the 'tower of Babylon' metaphor, of 'control systems managers' arrogantly trying to build the whole system from their preferred CS package by foisting their will on others, and watch the whole thing collapse into a pile of rubble.

4 ACKNOWLEDMENTS

The KGB/Cosylab team thanks the institutes DESY and the ORNL for hosting us and showing continuous support for our ideas.

REFERENCES

[1] http://kgb.ijs.si/KGB

[2] M. Kadunc et al., The Object Oriented Approach to Control Applications and Machine Physics Calculations with Java Technology, ICALEPCS01, San Jose 2001
I.Verstovsek et al., The New Abeans and CosyBeans: Cutting Edge Application and User Interface Framework, PCaPAC02, Frascati 2002.