CAN THE ACCELERATOR CONTROL SYSTEM BE BOUGHT FROM INDUSTRY?

M.Plesko, Cosylab, Ljubljana, Slovenia.

Abstract

This presentation is intended for project leaders and specialists, whose components depend on the control system, which is nearly everybody apart from control experts. The presentation will explain the basic concepts of an accelerator control system, illustrate the similarities and differences among the most popular packages, which are nicely disguised in acronyms such as EPICS, TANGO, TINE, DOOCS, COACK, XAL, CDEV, etc. and compare them to commercial control systems (DCS and SCADA) and LabView. The second part of the presentation will analyse whether a control system is in principle a component as any other and whether therefore in principle it should be bought eventually from a competent supplier like all the other components. It will identify the reasons why many people are reluctant to outsource control systems and illustrate this with some personal experiences and suggestions how to overcome these problems. The talk will conclude by showing how naively we have started a spin-off company [1] to commercialize the accelerator control system that we have developed, how we have found sustainable sources of business, and how we see the future in this and related markets.

INTRODUCTION

Let's start with a philosophical statement, which will appear indirectly throughout all this paper: The control system (CS) is not only the "glue" that keeps the equipment together. It is also a model of, or better, a container of the processes going on in the accelerator. It differs from other equipment exactly in that sense. Other, physical pieces of equipment have their own separate existence, while the CS is nothing without equipment and yet it appears to give "life" the equipment its raison d'etre, because it makes it accessible. This fundamental difference and the fact that software is considered immaterial are, in my own opinion, the two reasons why the CS was able to keep such a special position in all accelerator projects I had the opportunity to work for or with.

Therefore it is important to remember that the expectations of the final user, which is mainly the operator and only sometimes the engineer, should ultimately determine the design of the CS. On one end of the CS she/he requires the physical devices to be added easily and on the other she/he wants a powerful and easy-to-use user-interface. No matter the technology and no matter all those incomprehensible TLAs (TLA is a three letter acronym that stands for "three letter acronym"). After all, it worked perfectly well with cables and mechanical sliders and gauges half a century ago, didn't it?

THE BASIC CONCEPTS OF AN ACCELERATOR CONTROL SYSTEM

The Basic Architecture

It would be relatively easy with today's technology to read one value at one end and send it to the operator screen. But in addition, one has to take into account that specifications are often modified during the course of development, usually by the addition of features or even physical components. All these requirements demand a CS that is able to provide a great deal of flexibility.

Unlimited flexibility, however, results in unacceptably high cost. In order to achieve reasonable flexibility at low cost, one designs a small number of fundamental building blocks across the whole control system, both in hardware and software that are not allowed to be altered in any way. The design of the software blocks is usually associated with one of the CS packages, such as EPICS, DOOCS, TANGO, TINE, etc. The hardware blocks are individual boards. Ideally this design should be frozen only after carefully investigating the available requirements and after having predicted possible future needs.

In reality, we usually work backwards. We decide on a control system package, then we decide on a given hardware technology (say, VME, compact PCI, etc.) and then we match our specifications to that. If we have specifications at all at this stage. But that is a subject of a section further below.

One may wonder why this approach works. The simple reason is that nowadays, practically any technology is capable of doing the job. It may need slightly faster computers, it may be vastly more expensive to implement in terms of money or time, but as CS are usually not on the critical path of a project and cost less than the major components, nobody really notices this. So let's just keep in mind that there is room for improvement. On the other hand, this improvement may be smaller than the gain of just copying whatever concept the most recent project has adopted. If it worked for them, …

Now we have to put the blocks into a nice architecture. The accelerator is composed of physically distributed components. This means that the part of the control system that is related to the equipment (data-taking, input/output, or whatever we call it) is separated from the part, which displays values to the operator. So we have at least two separate layers or tiers. We see already at this early stage the arbitrariness of choosing names. This is one of the aspects that make CS so confusing. Especially to physicists and engineers, who go through a long drill to remember the exact differences of words that are otherwise synonyms. After all, which layman would differ between force, power and strength? Computer people

apparently don't either and to make matters worse, invent every few years a set of completely new expressions.

Just to illustrate this in a few examples, let's compare modular programming "speak" to object oriented (OO) "speak". Would you know that in OO, "persistent store" means saving data to a file and that "methods" are the same as functions and procedures, which in turn replaced the good old subroutines! Oh yes, and what were once variables are called "fields" in OO programming.

Let's go back to our two tiers. We have thus, to simplify, one computer at the device and one in the control room. Obviously, the cheapest way of connecting them is via a local area network – LAN. LAN is a simple concept, like the phone, that hides a lot of technology from the casual user, like a phone system does, too. And it works very similarly to a private phone exchange. Essentially, a LAN nowadays uses so-called UTP (unshielded twisted pair) cables, over which an electrical protocol called Ethernet is operating. Data are transmitted digitally in packets and the rules how those packets are assembled, checked for arrival and resubmitted, are described with the acronym TCP/IP. It is an essential part of the Internet protocols, therefore nowadays anybody can say that they use Internet technologies, if they connect two PCs with a cable.

To finish our simplified example, this is is the essential architecture of a control system:

- Computers, which are attached to devices and read/write data from/to them
- Computers, which display those values to the operator and accept commands from her/him
- A network with a communication protocol that connects all those computer with each other, just like the internet connects all computers in the world.

Two Tiers Versus Three Tiers

You can skip the following paragraph in order to avoid further confusion. But if you want to know why there are also control systems with three and four tiers read on:

Before the times of ubiquitous 100 Mbit Ethernet (still remember the famous coax yellow cable?) and cheap processors, CS designers had to save bandwidth and money with the following trick: They did not have all computers with Ethernet and TCP/IP. Instead, they had several low performance (sometimes even without an operating system) computers connected to devices.

Then they had a cable connecting those computers with a normal PC-type computer (it wasn't a PC then, but that's not important for the argument). Such a cable is something like a logical extension of the PC. As a PC has an internal bus that allows to add several cards, such cable with the corresponding protocol is called a fieldbus. Opposed to the internal bus, which is normally on a backplane, is goes out of the PC and into the "field". Well known fieldbuses are for example CAMAC, a very old standard developed for nuclear physics in the times when 8-bit computers were the state of the art. Modern fieldbuses are CAN or Profibus (of which there are two versions, but only DP is used nowadays).

Also CAN has several versions and is called DeviceNet in the USA when sold by Allan-Bradley. There are other versions of CAN, because people wanted to add addition functionality and frameworks for protocol handling, just like TCP/IP sits on top of Ethernet. This is all very friendly to developers, but creates confusion for outside people and for those, who try for the first time to understand the field of control systems and are expected to decide on a technology.

So now, we have three tiers:

- The dumb computers (often called controllers)
- The computers that connect those dumb ones – there are usually several disjoint branches
- The computers for the operator, often called operator consoles or just consoles.

The computer in the middle have only the function to serve data to the consoles, therefore they are often called servers.

Now we have also two different types of communication: a fieldbus and the well known Ethernet with TCP/IP. Obviously, this subdivision adds to the complexity and what is even worse, it introduces new technologies that have to be learned, managed and are another risk. As often in IT, the greatest risk is that technology becomes obsolescent – suddenly the new software does not work with the old hardware or vice versa, or the fieldbus is not supported in the new operating system, etc.

The only way to minimize this risk is to reduce the number of technologies – spreading out NEVER reduces the risk. Therefore my suggestion is, and I have developed control systems with fieldbuses before, place your bets on the ubiquity, low-cost and availability of Ethernet and go for a two tier architecture.

Just for completeness sake, there are also four tier architectures, double Ethernet networks for redundancy. Unless there are really good reasons, bring in redundancy in a way which keeps the two-tier architecture. And if you really want more tiers, use software-tiers: there, it is a matter of definition what a tier is and not a matter of different technologies.

Interfaces, i.e. the Contract

In real life, there are more complex functions required than just reading values and applying commands. We must therefore refine the architecture by further splitting each tier into 3-4 independent layers, interconnected via interfaces. The implementation of solutions for different cases in the layers became the above mentioned building blocks.

This structure allowed the programmers to develop the layers independently of each other. Ideally, the only constraints to the programmer are the interfaces that had been carefully designed in the first step of the design process in order to minimize interdependencies between layers, both in code and data. These interfaces define all the possible interactions between layers in a consistent

way. The result is cleaner code and better equipment, since every participant in the development process only has a limited number of things to worry about. If some components have to be optimised they can just be rebuilt from scratch, without affecting other components, because the interfaces stay the same. In addition, testing, debugging and error correction is much easier, as it can be kept localized.

We see that communication through standard protocols (such as channel access in EPICS) is not enough. Clean and consistent interfaces must be designed and agreed above these protocols to provide a suitable context. Those interfaces are just like contracts – one programmer can be sure to expect exactly the right data in the right way from the other programmer. In case something doesn't work it can be tested unambiguously, where the problem lies. Although often both programmers must work together to find the causes in the quickest way. Here, as in business life, just claiming that one is right and bringing out paragraphs of "the contract" is legally possible, but is not productive and creates bad resentments. The philosophy of our company on in software projects is, even when we work as business partners, or as subcontractors and we have double checked that the problems are not due to us, we offer help to the partner in finding the bugs, sometimes even at our cost.

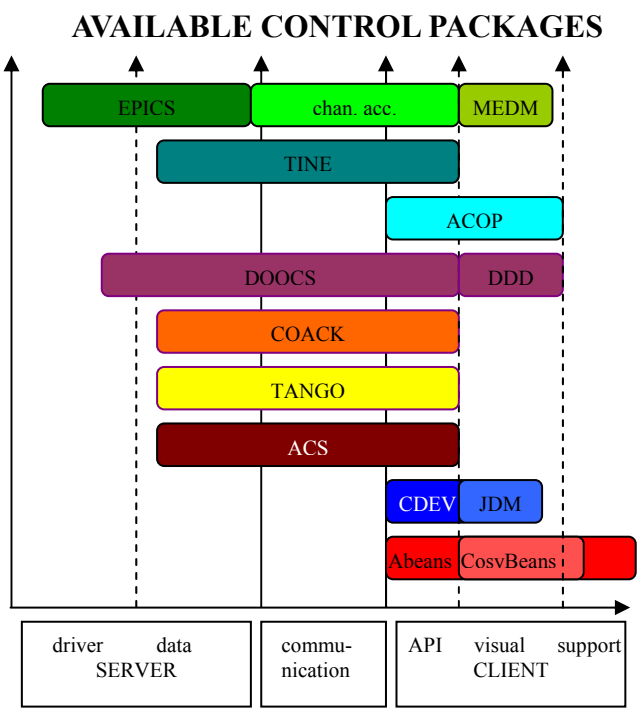

Figure 2: A comparison of control system packages and the layers they cover.

There are several competing free or open-source control system (CS) packages and individual components that have been developed at accelerator labs and are now being shared more or less successfully. Many 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. For the sake of example we will be mentioning only some systems, which does not represent an endorsement by the authors, nor is it any reflection on anybody else's system. We will also not further discuss XAL and Databush, which are packages for machine physics calculations and should be compared to the more and more popular Matlab machine physics libraries.

The different coverage of control system packages is shown in figure 1. 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 [2] for a full comparison and allude to certain aspects below.

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 new 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.

For commercial packages, there are several terms used in different occasions, such as industrial control systems, commercial control systems, SCADA (supervisory control and data acquisition) or DCS (distributed controls systems) and often people just use the terms to distinguish them from control systems that have grown in our community and are free. Again the names are different for historic reasons are really mean one and the same thing Maybe the biggest difference among those two groups is who the various systems are aimed at. Industrial systems are aimed at people who just want to concentrate on the application, with as little programming – often preferably none - as possible, while free systems are aimed at the people who need flexibility over anything else.

When control people (including myself) will want to convince you to adapt to their control system package of choice, be prepared for a barrage of buzzwords. In reality it matters little, which system they choose. Rather ask them for a list of actions and clear development procedures. Don't forget to ask for a test plan and documentation **before** the implementation starts, although they will tell you this can not be done. If you paid an outside company and they wouldn't get the money before

all is finished, it would become not only possible, but standard industry practice.

WHAT REALLY IS IMPORTANT ABOUT CONTROL SYSTEMS

All the "sexy" technology lets us often forget that control system is an engineering discipline like all the others, but but with an even more complicated development cycle:

- Write specifications
- Architecture
- Design
- Prototyping – probably the only fun part
- Define test procedures
- Implementation (coding) – the only software part
- Writing documentation
- Testing (follow ISO procedures)
- Debugging
- Acceptance at customer

Don't forget, that even in-house control groups have a customer – physicists and operators, which must be involved in the specification, testing and acceptance phases.

Think like this: in vacuum, a specific tube or chamber is just the result of much designing before and testing after manufacturing. So is programming and running programs just a small step in the whole process – or so it should be. Often, programming is considered the key and only aspect apart from buying some hardware. The simple reason for this is that anybody can design and write at least simple programs, but not anybody can work with tools. Not to become philosophical, we conclude that control systems are the closest to pure procedures, which our mind seems perfectly adapted for and allows at least in principle to reverse any mistake at no apparent cost. The true cost, indeed, is very high – lost time, which in our modern society becomes more and more the most precious commodity of all.

What A Project Leader Must Look for in CS

The nearly religious discussions about all those nice features individual control system packages have (see next section for an overview of them) more often than not obscures the items which a project leader really must take care of. Although not strictly part of the control system, they fall into the domain of the control system group and in reality form the largest part of their work:

- Signal list:. Some call it the golden or master list. Although it is so common sense that you laugh at me now, I have yet to see a project where the signal list has not been completed in the last minute or actually after some of the development has been done. The signal list really is a contract between the equipment specialist, the control expert and the operator. A contract that should be honored to the maximum extent. Changing something only because it is easy to change can have serious consequences later.
- Signal names and general name conventions: a part of the signal list, but too obvious to be taken seriously and therefore often neglected. But the moment more than one person is involved, names must be unique and a good naming convention helps to keep it that way.
- Alarm levels and operation limits: often left empty, because even the device expert does not know reasonably acceptable operation limits. This is later forgotten and only rediscovered many years into operation
- Configuration management: having procedures in place that deal with changing signal list, changing hardware, and changing software in such a way that all interdependencies are taken care of and that one number must not be changed in several places (otherwise it won't be changes and the whole CS becomes inconsistent)
- Logistics of installations: equipment can't be tested without CS, CS can't be tested without equipment – people often forget that although only careful planning is needed. involving both sides, the CS people and the device experts.
- Bugs: To err is human, but for real crap, you need a computer. Seriously: it is normal that bugs happen, because the complexity of the software is just too great. One has to plan a lot of time for testing and fixing bugs. And one has to live with workarounds. Having said that, the number of bugs and the cost they have can and must be minimized with strict control of the development process.

SHOULD THE CONTROL SYSTEM BE BOUGHT OR MADE IN-HOUSE?

The previous section clearly showed that the control system should be developed by engineers with experiences in software projects and not just by programmers, be they computer scientists or physicists. Most labs don't have people with such experience. Commercial system integrators do have this knowledge, but they don't understand accelerators, which is very important. After all, also big IT projects are awarded to companies with a proven list of references and not to general "programming companies". The petrol industry has a completely different set of software suppliers than the telecom industry. Even big system integrators like IBM have completely different departments dealing with different customers.

All the above should lead to the conclusion that a company like Cosylab that is specialized in control systems specifically for accelerators, telescopes and beamlines, should have good business.

Unfortunately, in the scientific and accelerator community there are many mutually exclusive preconceptions about why control systems should be

made in-house. Unfortunately for us as a company, we must fight them all on different places and occasions.

One standard answer is: "we have to consider many special cases and moving targets, so we don't know yet what work to give to you – we'll call you later when all is defined". False, but we never get the call: initially, it is too early, in the middle they don't have time to define clear tasks for us, and in the end they admit it would have been easier with our help, but now it's too late. Actually it is never too early and never too late.

The strongest argument is that outsiders will themselves define the work and make sure they deliver all they have promised, because you have the leverage not to pay them if you are not satisfied.

Another belief is that in-house people can fix problems or write a new program overnight, while outsiders can not. This may not always be the best way to work, because one skips the most important steps in the development phase described previously. But is has certain benefits to be able to make quick fixes fast, especially when the accelerator is standing still for some obscure bugs. Having understood these issues, we at Cosylab offer to labs a combination of in- and out-sourcing: we leave one person permanently at the lab to collect requests and be available for quick fixes. In parallel, we back him up by the large team at Cosylab to provide expertise on all possible aspects. If the lab is in a different continent, we literally fix problems over night.

So our new business model is something like a sale in a supermarket: Pay one, get many! The real benefit for a lab is that they get a quarter each of a designer, an implementer, a tester and a project manager. For them it is actually cheaper to pay one external than to hire four different experts.

To conclude this section, my argument is to not necessarily buy the complete control system, but to outsource much of the development and installation, including writing documentation, which nobody in the lab will do, while a small company like ours is happy do earn a living with it.

About twenty years ago, control electronics started to shift from in-house to commercial off-the-shelf and now control software could be in a similar transition phase.

HOW WE STARTED THE COMPANY

The team started as a group of students under my supervision at the J. Stefan Institute in Ljubljana, who got the contract to develop the control system for ANKA and part of the ALMA Common Software. Our team consisted practically only of undergraduate students. Stimulating and rewarding the students with cutting-edge technologies and travel to conferences like this and installation fieldwork are an important positive factor in raising their motivation. However, building any system with a group of inexperienced students is quite a challenging task. To cope with it, we had to use many software engineering tools: CVS for versioning and source archiving, Bugzilla for keeping our bugs in order, a to-do list for managing tasks, an activity log and also many other programs and scripts, some found on the Internet and some made by ourselves. In the end, we had to become organized like a professional company. Documentation and demos can be found on our old homepage [2].

So we had a veteran team with an average age of 22. The oldest members have already graduated and left and we would have lost all our investment in the people, if we had let this trend to continue. Our institute couldn't hire the whole team; therefore we just decided to create a spin-off company for developing and installing control systems for accelerators and other large experimental facilities [1], with the full blessing of our institute director and division leader. True to the research community that we grew in, the vision of the company is to make a living with our work instead of selling software licenses. And true to our philosophy of high motivation, all initial employees are co-owners.

An interesting fact is that our initial financial plan for the first four years was practically completely correct in predicted turnover. However, the customer that we eventually had were almost completely different from those we have foreseen. We have also substantial activities in completely other markets such as Geographical Information Systems (GIS), telecom and automotive electronics, where we re-use the technology we have developed for accelerators. It is important to always be looking for novel business opportunities

CONCLUSIONS

Now, we are the leading commercial provider specialized in accelerator and beamline control systems. The company has grown to over 25 employees, but we still work with students of physics, electronics, software science and mathematics, of which we have about 50 in our so-called CosyAcademy pipeline..

Among our customers are over 20 major accelerator labs all around the world and companies that supply equipment to accelerators such as Bergoz, Danfysik, FMB, Instrumentation Technologies and Oxford Danfysik. We will probably never get rich, but we do a good and competent job and are widely respected for this. If we make mistakes, we admit them, apologize and invest all efforts to fix them.

So we are a proof that the simple answer to the title of this paper is "yes". The more complicated and realistic answer is, "yes, but you must first choose the right company, one with good understanding of accelerators and with proven competence. Then look what your people can do best and leave the rest to outsourcing".

REFERENCES

[1] www.cosylab.com.
[2] http://kgb.ijs.si/KGB