Control Sheet

Cosylab's newsletter

VOLUME 4 04/06/2009

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

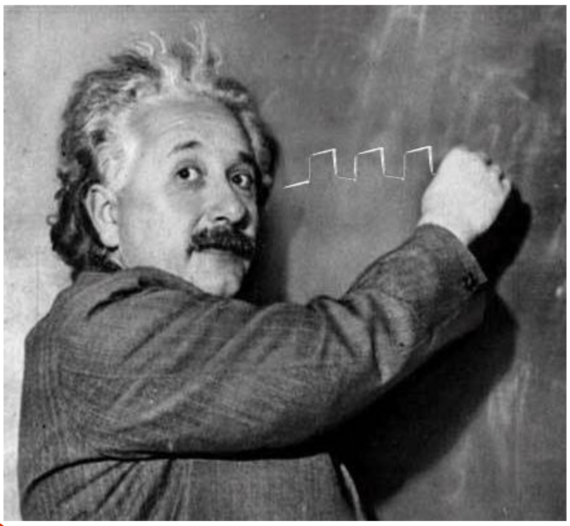
BY: MARK PLESKO

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 might not 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 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.



The strongest argument against this misconception 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 cannot. This may not always be the best way to work, because one skips the most important steps in the development phase. 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 on in a different continent, we literally fix problems over night.

To conclude this article, 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 to 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 TO GET A FREE COSYLAB T-SHIRT?

SEND US AN INTERESTING STORY AND GET A T-SHIRT.

IF YOU ARE ALREADY A PROUD OWNER OF THE POPULAR COSYLAB T-SHIRT, THEN YOU CAN ALSO CHOOSE BETWEEN A VINTAGE KGB T-SHIRT, SLOVENIAN CHOCOLATE OR COSYLAB'S USB COFFEE CUP WARMER WITH MUG (EPICS DRIVER NOT INCLUDED!).

Middleware Evaluation

BY: KLEMEN ŽAGAR

ITER Organization [1] contracted Cosylab to evaluate middleware technologies that would be suitable for implementing ITER's distributed control system. Together with ITER, we have decided to take a look at the following middleware technologies:

- EPICS' *Channel Access* [2] which is an integral part of the EPICS control system, but could be used independently as well,
- *OmniORB* CORBA [3], which is used in (among others) as the middleware for the TANGO and CERN's LHC control systems,
- ZeroC's *Internet Communications Engine* (Ice) [4], which is considered a modern alternative to CORBA,
- and a commercial implementation of the Object Management Group's *Data Distribution Service* (DDS, [5]) standard, which provides efficient means for distribution of real-time data.

We assessed applicability of these middleware technologies against the ITER control system's requirements (data streaming, remote command invocation, reliability, etc.), and measured throughput and latency of data transmission that the technologies allow.

From perspective of functionality all technologies fit the bill, though some might be more cumbersome than others for certain applications. For example, to implement a remote-procedure-call style communication in EPICS, one would need to manually craft the code to marshal and unmarshal the parameters and return values, which in case of CORBA, Ice and DDS are generated from interface definitions.

In terms of performance, all technologies achieve benchmarks of the same order of magnitude (see figures). We were positively surprised with *Channel Access* and *OmniORB* – despite their relative age, they remain sharp performance-wise. In our tests, *Channel Access* was even at a disadvantage that we have used it together with EPICS database processing to realize a kind of notification service – the performance of EPICS-less *Channel Access* would likely be even better.

We have analyzed scalability by reviewing third-party benchmarks. We have found that DDS is the leader of the pack, as it efficiently leverages IP multicasting. With EPICS, the system is still very scalable as the load can be transparently balanced across a large number of computing nodes, but is less scalable than DDS as it relies on the point-to-point TCP protocol for data transfer. With Ice and CORBA, scalability depends on how exactly the technologies are applied, but straightforward application of non-federated centralized notification services might quickly make these services a bottleneck and a single point of failure.

[1] The International Thermonuclear Experimental Reactor (ITER), http://www.iter.org

[2] Experimental Physics and Industrial Control System (EPICS), http://aps.anl.gov/epics/

[3] OmniORB CORBA, http://omniorb.sourceforge.net/

[4] ZeroC Ice, http://www.zeroc.com/

[5] Object Management Group: Data Distribution Service, http://portals.omg.org/dds

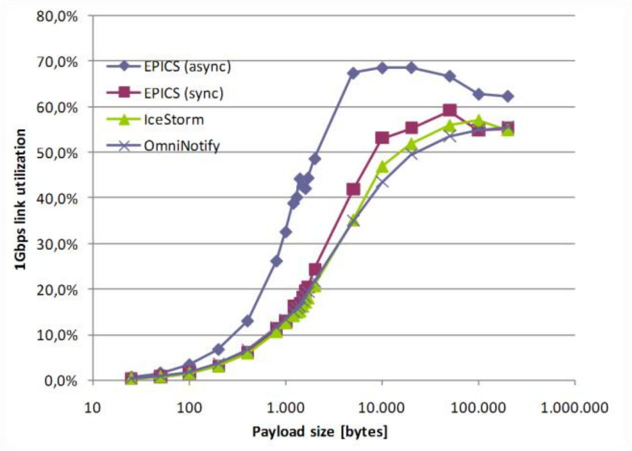


Figure 1: utilization of network bandwidth (compared to raw UDP throughput) of middleware solutions. The figure compares scenarios where the stream of data needs to traverse a "notification service" (EPICS database or a notification service).

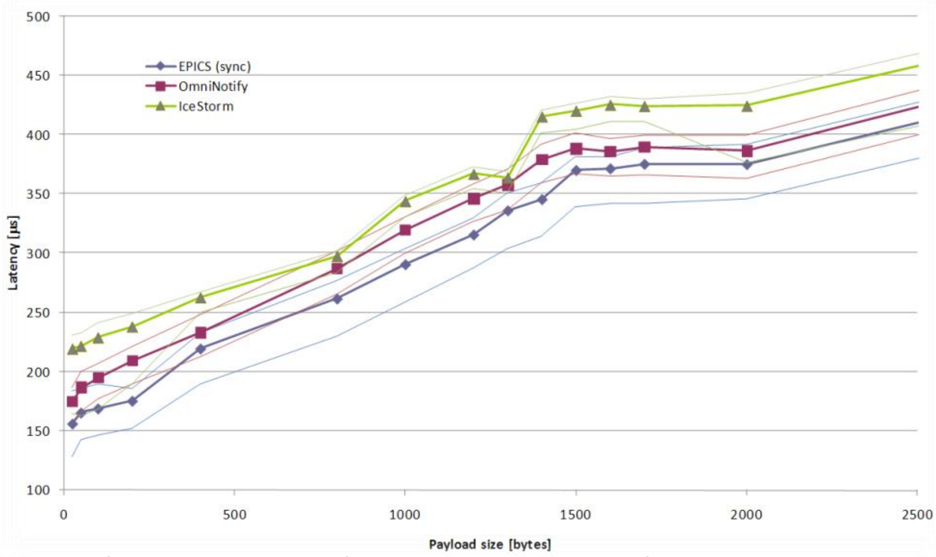


Figure 2: round-trip latency of middleware solutions, The figure compares scenarios where the stream of data needs to traverse a "notification service" (EPICS database or a notification service).

Extreme power and extreme lure of FPGA programming

BY: JOŽE DEDIČ

FPGA development might show similar coding paradigm to SW development, but it is based on rudimentarily different concepts. Both share some disciplines but they are based on completely different paradigms. The FPGA system can typically offer much more than the software solution; it can have greater flexibility and performance, but you have to exercise larger set of skills to meet your requirements. Hand in hand with extreme power FPGAs offer there is also an extreme lure: they make people think that anybody with a little programming experience can develop complex digital electronics. In a way, they are similar to sports cars; the more power they have, the less forgiving they are to even small mistakes. And just like sports cars, FPGAs are not meant for the regular programmer (driver) to develop fast digital electronics (drive races), but a tool for experienced electronics developers.

Software = paradigm of sequential processing

A couple of decades ago transistors, as basic building blocks of integrated circuits, were very expensive and everything had to be done with as few of them as possible. Engineers invented processor; a device that sacrifices speed (sequentially executing a basic set of actions) for cost (basic actions = lower transistor count). Accomplishing any complex task merely means putting those actions in correct order, i.e. software programming. Nowadays the execution speed is several orders of magnitude higher and the instruction set much larger, but the paradigm of sequential processing stayed the same.

FPGA development = paradigm of hardware description

As integration-circuit industry evolved, hardware became orders of magnitudes more affordable and enormously large configurable circuits surfaced on the market, known as field programmable gate array or **FPGA** circuits. FPGA circuits consist of a multitude of basic building blocks; developer has the ability to configure functionality of every building block (for any action) and can choose how to connect them. There are millions of blocks waiting to be configured, all of them allowing to be used in parallel.

It's 6:1 for SW when it comes to development-tools maturity

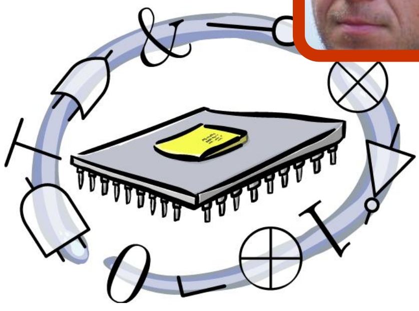
Processors exist for 6 decades now, FPGA circuits only 1; the former tools are much more mature. With FPGA tools you should be accustomed finding bugs or using workarounds. What strikes me all over again, is the absence of small productivity enhancements; decent versioning control support or code completion that SW programmers can benefit from. In FPGA world it is often up to the developer's ingenuity.

FPGA design is inherently an order of magnitude more complex than SW design

Besides pondering processing with massive parallelism and keeping in mind hardware concepts like registers, pipelining, clock cycles, and communication between asynchronous modules, the development techniques are significantly different. SW is fundamentally sequential. In FPGA everything is parallel, from basic blocks to the IP cores of specialized functionality. In SW you only have to define what a fixed hardware does in time, in FPGA you need to first define what hardware is (remember, it can mimic anything you need).

FPGA debugging can be extremely challenging, experience is key

You need a significant amount of time to compile a design to hardware code and run it in the FPGA, and it normally takes multiple compiles to debug the code. Furthermore, even when you do wait for the compile process to finish, it is difficult to probe into the functionality of the FPGA logic because there is no concept of typical software debugging as you are dealing with signals, not values: in SW, a zero is a zero and a one is a one, unmistakably. In FPGA circuits there are hold and setup times, etc. all of which have to be taken into account not only in the design of the electronics board, but also in the programming of the FPGA in particular when crossing clock-domains (sections of design, running on different clocks). In SW you don't have to deal with meta-stable states and glitches; in FPGA it is very difficult to track and remove odd and rare bugs, especially if they originate from poor coding style or violating any of the rules, e.g. crossing clock-domain.



Testing and verification are crucial, but too often avoided

"As a rule of thumb you need 1 person to test the code written by 4 SW developers. For the FPGA development the ratio should be 1:1". There is no straightforward answer on how to test FPGA (sub)modules and it is often a conceptual challenge how to verify required behavior. There are numerous software packages and language extensions available for this reason (they can be pricey) but they all require in-depth understanding of the verification concepts. Even once you know how to do it, writing verification test-benches is still time-consuming. Knowing (or not knowing) this, developers avoid systematic verification, taking the standpoint "show me a bug and I'll fix it".

Typical FPGA developers don't embrace best SW development practices

Firstly, tools are more heterogeneous and secondly, FPGA designers typically lack the training on best coding etiquette; both collaboration aspects as well as coding style. Issues arising from poor coding style are very hard to debug. Sadly, too often FPGA programming could not be reproduced without developer's laptop. And it might be even worse if you can not reproduce a binary bitstream of a running version.

Bottom line is, FPGAs are here and we'll use them more and more

They can be used to achieve practically anything. But be aware, compared to SW, FPGA discipline is still young and conceptually more difficult to master – see difference in paradigm. FPGA development requires control over typical SW development pitfalls plus the upgrade to better organizational skills. No matter how easy the initial example is, the learning curve becomes very steep. As with SW development, FPGA development requires mastering of multiple disciplines and coding is just one of them. . May your FPGA code live long and prosper!

Control SH!T: GTD free

BY: IGOR KRIŽNAR

GTD in this article stands for Getting Things Done® a popular productivity methodology written down by David Allen (see David Allen & Co. official site [1]. If you haven't heard about this, it is certainly something worth checking out. There are a lot of resources about GTD on Internet. "Just Google for GTD."

When I came across the GTD it made a huge impact on me and I wanted to try it out immediately. I have bought the electronic labeler, reorganized my stack of papers, set up in-baskets and 42 folders. Some of the tricks worked better, some I have abandoned after a while. It was fun and made me feel more organized and productive. But I was never really satisfied with the so called "reliable list of next actions" which is a key requirement to make the system work. My first attempt was made with simple outline-fashioned word processing document. It was fine for a while but when it grew beyond certain size it became progressively unmanageable. It was time to change to a real task management program.

Project management and tracking systems like Request Tracker or Jira or Mantis are fine for what they are made for, i.e. for managing projects. But they are clumsy to use when managing a large number of short actions, which deal with personal and professional day-to-day life. So I have tried several personal to-do managers. None of those that I tried was acceptable, or they did not support GTD concepts enough or they could not beat my outline word document when it comes to simplicity.

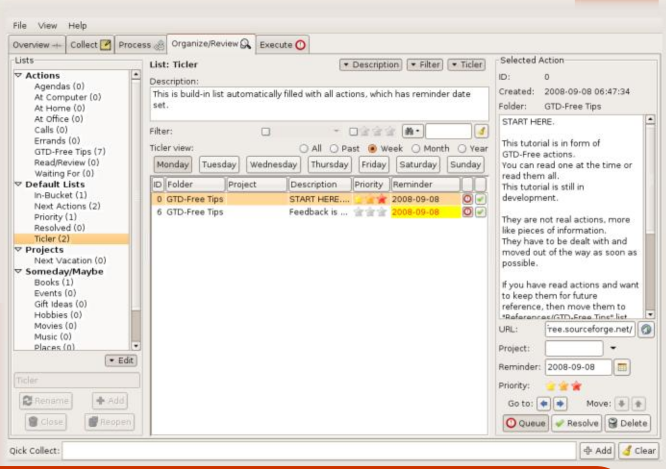
About that time it became apparent that GTD was not only reliable and stimulating routine, but in fact it did help to improve my productivity in professional and personal life. I was able to divert some of my time to my personal geek-ish wish: to run my own project at Sourceforge.net. So GTD [2] project was created, a personal TODO/action manager. I've tried to keep simplicity of an outline document and add features, which would help enhance GTD workflow routine.

After a year GTD-Free is still a work in progress. A lot of things have been done but many more are on the wish list. The things that are done are working well and stable. Besides me many other people seems to use it for managing their day-to-day activities. By the comment on the forums [3] they seem to love it, check [2] and see for yourselves. My free time gained by productivity gain is spent on GTD-Free, so I'm back at my usual level of stress, but I guess I love it just the way it is.

[1] David Allen & Co. Offical site, http://www.davidco.com/

[2] GTD project, http://gtd-free.sourceforge.net

[3] GTD forum, http://sourceforge.net/forum/?group_id=219074



List: Ticler

Description:
This is build-in list automatically filled with all actions, which has reminder date set.

Filter:

Ticler view: [✓ Week]

ID	Folder	Project	Description	Priority	Reminder
0	GTD-Free Tips		START HERE....	★★★	2008-09-08
6	GTD-Free Tips		Feedback is ...	☆☆☆	2008-09-08

Selected Action
ID: 0
Created: 2008-09-08 06:47:34
Folder: GTD-Free Tips

START HERE.

This tutorial is in form of GTD-Free actions.
You can read one at the time or read them all.
This tutorial is still in development.

They are not real actions, more like pieces of information.
They have to be dealt with and moved out of the way as soon as possible.

If you have read actions and want to keep them for future reference, then move them to "References/GTD-Free Tips" list

URL: ree.sourceforge.net/
Project:
Reminder: 2008-09-08
Priority: ★★★

Qick Collect:

Cosylab's Hall of Fame

Dear tech-talk devotees,

I downloaded the latest JCA and CAJ from the Cosyab site, then found the example channel access server program (BasicExample.java in the test/..../cas/test directory).
Made minor mods and got it working in about 5 mins. A far cry from my first attempt many years ago to use the C++ PCAS library.

Many thanks, and my hat is off to all of you at Cosylab. I may never write a C++ PCAS program again!
Sincerely,

Elliott

COSYLAB d.d.
Teslova ulica 30
SI-1000 Ljubljana
Slovenia

Phone: +38614776676
Fax: +38614776610
E-mail: controlsheet@cosylab.com
Web: www.cosylab.com

visit us on http://cosylab.com/solutions/particle_accelerators/control_sheet/