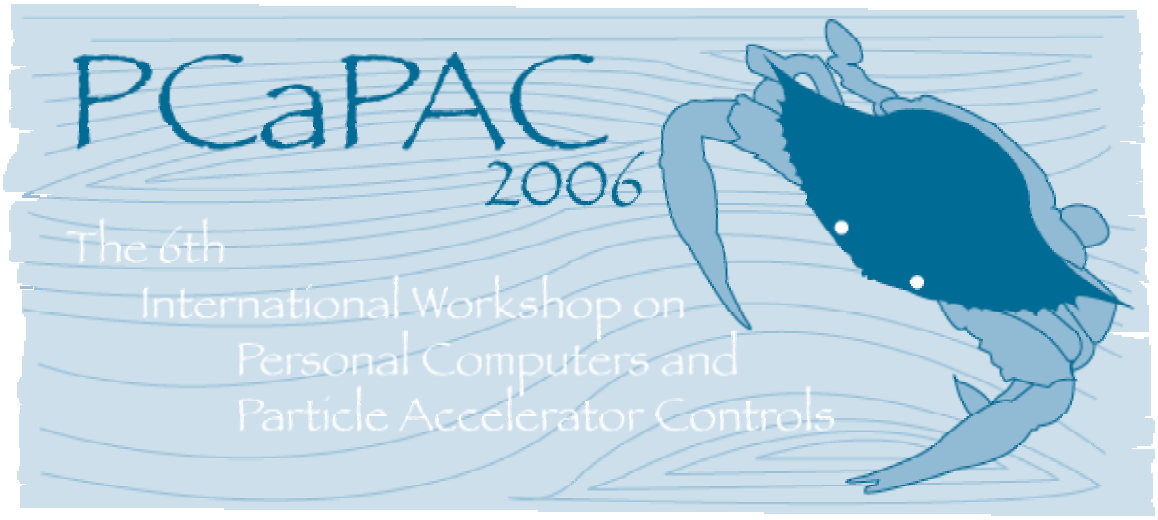


24-27 October 2006
Jefferson Lab
Newport News, VA, USA

PROCEEDINGS

JLAB-ACO-07-628

Editors: M. Bickley and P. Chevtsov

CONTENTS

Preface ……. viii
PCaPAC 2006 schedule and some workshop photos …..……..… x

CONTRIBUTED PAPERS

TANGO Control System Status ……...…… 3
J.-M. Chaize

The New Control System for the Future Low-Emittance Light Source PETRA-3 at DESY: From Conceptual Design Work to Realization …… 7
R. Bacher

Status of the SCSS Prototype Accelerator and Control system …….. 11
T. Ohata, T. Fukui, T. Hirono, N. Hosoda, T. Masuda, T. Matsushita, T. Ohshima, M. Takeuchi, R.Tanaka, A. Yamashita, M. Kitamura, H. Maesaka, and Y. Otake

Migrating the STAR Slow Controls System to PC's ……... 15
J. Fujita, Y. Gorbunov, M. Cherney, W. Waggoner, J. Burns, M. Brnicky, and R. Thomen

Using the Common Device Interface in TINE …… 17
P. Duval and H. Wu

The Interconnection of TINE and STARS …… 20
T. Kosuge, P. Duval, Y. Nagatani, and K. Nigorikawa

Embedding a TANGO Device into a Digital BPM …… 23
C. Scafuri, V. Forchi, G. Gaio, and N. Leclercq

Control System for the FFAG Complex in KURRI …….. 26
M. Tanigaki, K. Takamiya, H. Yoshino, N. Abe, T. Takeshita, Y. Mori, K. Mishima, S. Shiroya, Y. Kijima, and M. Ikeda

Operational Experience with Synchrotron Light Interferometers for CEBAF Experimental Beam Lines …….. 30
P. Chevtsov

Beyond PCs: Accelerator Controls on Programmable Logic ……..… 33
M. Plesko, K. Zagar, and A. Hasanovic

Mono for Cross-Platform Control System Environment ……..…….. 37
H. Nishimura and C. Timossi

A Configurable Interlock System for RF-Stations at XFEL ……... 39
M. Penno, T. Grevsmuehl, H. Leich, A. Kretzschmann, W. Koehler,
B. Petrosyan, G. Trowitzsch, and R. Wenndorff

Magnetic Field Mapping (MFM) System for Super Conducting Cyclotron (SCC) in VECC ……..…… 40
S. Pal, A. Roy, T. Bhattacharjee, N. Chaddha,
R. B. Bhole, and S. Dasgupta

MCS-8 Eight Axis Embedded Motion Control System ……..…….. 43
G. Jansa, R. Gajsek, and M. Kobal

MicroIOC: PC and Control System Longevity ……..……... 49
A. Podborsek, D. Golob, A. Hasanovic, I. Kriznar,
R. Sabjan, and M. Plesko

Ethernet-based Fieldbus Functionality for Neutron Scattering Experiments with PROFINET IO ……..……..... 53
H. Kleines, S. Detert, F. Suxdorf, and M. Drochner

EPICS SCA Clients on the .NET x64 Platform ……..……...…... 56
C. Timossi and H. Nishimura

A Tutorial on Project Management ……..……...…... 58
J. Kamenik, P. Kolaric, M. Plesko, and I. Verstovsek

Standardisation of the PSI Accelerator Control System ……...…... 61
D. Anicic, T. Korhonen, A.C. Mezger, and D. Vermeulen

Status of ISAC Control System ……..……..…..…... 63
C. Payne

First Operation with SPARC Control System ……...…... 66
M. Bellaveglia, G. Di Pirro, D. Filippetto, E. Pace,
L. Catani, and A. Cianchi

EPICS ArchiveViewer Project Status ……..……..…...…... 69
S. Chevtsov

Device Address Redirection as a Tool in the TINE Control System ……...……..…...…... 72
S. Herb and P. Duval

Web GUI for the TANGO Control System ……..……..…...…... 75
L. Zambon and M. Lonza

A Communication Protocol for a Distributed Control System with LabVIEW ……..……..…...…... 78
L. Catani

User Requirements for the PETRA-3 Control System at DESY ……...……..….……..…... 82
M. Bieler, A. Brinkmann, and U. Zobjack

Off-Line Analysis Goes On-Line! ……..……..…...…... 84
M. Lomperski

A Users Perspective ……..… ……..……..…...…... 86
I. Carlino

A Prototype of a Beam Steering Assistant Tool for Accelerator Operations ……..……..……...…... 90
M. Bickley and P. Chevtsov

Applications of Interest: A Relational Database Approach to Managing Control System Software Applications ……....…...…... 93
D. Quock, N. Arnold, D. Dohan, J. Anderson, and D. Clemons

Accelerator Management with Web-based GIS …….……...…... 96
A. Yamashita, Y. Ishizawa, T. Ohata, and M. Takeuchi

Status of the CEBAF Control System at JLAB …….....…... 99
M. Bickley

Control System Studio (CSS) ……..……...….... 102
M. Clausen, J. Hatje, and M. Möller

Automatically Configured Control System Using Compact PCI System ……..……..……...……...…..... 105
Y. Furukawa and T. Ohata

The Common Firmware Programming Interface for Fieldbus Related Projects at PETRA III ……...……...…..... 108
P. Bartkewicz and S. Herb

UI-Oriented Approach for Building Modular Control Programs in VEPP-5 Control System ……...……...…..... 111
D. Bolkhovityanov

A Device Server Generator for Control Systems ……..…….....….. 114
J. Wilgen and P. Duval

LivEPICS – An EPICS Linux Live CD for Small Applications, Training and Fly Tests ……..……..…...…..... 116
M. Giacchini and G. Bassato

Fault-tolerant EPICS Directory Service ……..…….……...….. 117
I. Habjan, K. Zagar, and M. Sekoranja

Java Device API and CosyBeans in the GSI Controls System ……..…….....….. 120
G. Froehlich, K. Hoeppner, U. Krause, V.R.W. Schaa, I. Kriznar, M. Plesko, and J. Bobnar

PC-based Innovations in the VEPP-4 Obsolete Control System ……..…….……...…….. 123
V. Kaplin, S. Karnaev, O. Meshkov, I. Morozov, O. Plotnikova, V. Smaluk, and A. Zhuravlev

Building and Deploying Loosely Coupled Console Applications ……..…….……..…..... 126
A. Labudda

RF System High Power Amplifier Software Conversion at Jefferson Lab……..…….……..... 129
G. Lahti, H. Dong, and T. Seeberger

Advantages of the Program-Based Logbook Submission GUI at Jefferson Lab ….…….…….…….... 131
T. McGuckin

Management in Temperature of RF Cavities of VEPP-4M Electron-Positron Facility …….…….... 134
E. Miginskaya, I. Morozov, V. Tsukanov, and A. Volkov

SNS IOCs Use of Relational Database to Supply Configuration File …..…….…….…….…….... 136
J.D. Purcell, W. Blokland, A. Liyu, J. Patton,
T. Pelaia, and A. Zhukov

Control System Using FL-net for Communication Between Different PLC …….…….... 139
A. Osanai

Compact Monitoring and Control System with Event Simulating …….…….…….... 142
V. Vinogradov

New High-Performance Modular Computer System Architectures for Control Network Applications ……...…….... 145
V. Vinogradov

The ACOP Family of Beans …….…….... 149
P. Duval, H. Wu, and I. Kriznar

ACS – An Open Source Control System Infrastructure …...…….... 152
K. Zagar, G. Chiozzi, M. Sekoranja, H. Sommer, M. Plesko,
B. Jeram, A. Caproni, R. Cirami, and P. Di Marcantonio

APPENDICES

List of Participants …...……..……... 155
Author Index ……..……... 162

PREFACE

The 6-th international PCaPAC (Personal Computers and Particle Accelerator Controls) workshop was hosted by Jefferson Lab, Newport News, Virginia, USA, from October 24-27, 2006. The workshop was attended by about eighty participants who represented accelerator control system developers of twenty four research institutes and centers from ten countries.

The main objectives of the conference were to discuss the most important issues of the use of PCs and modern IT technologies for controls of accelerators and to give scientists, engineers, and technicians a forum to exchange ideas on control problems and their solutions.

The workshop consisted of plenary sessions and poster sessions. No parallel sessions were held. Seventy seven oral and poster presentations were made during the conference, on the basis of which more than fifty papers were submitted by the authors and included in these Proceedings.

Many people contributed to the success of the workshop. Members of the Local Organizing Committee are listed below. M. Hightower, C. Lockwood, R. Bizot, and N. Vermeire from Jefferson Lab Director's office did a fabulous job in organizing and coordinating of all our efforts. Special mentions go to S. Kyte and her team for the development and support of the official PCaPAC 2006 web site, F. Dylla for his remarkable talk at the workshop dinner about vacuum and its importance for science and technology, C. Watson for his very nice presentation of supercomputing research at Jefferson Lab, D. Neal for his wonderful talk about "living on the web", M. Plesko and L. Catani for their excellent PCaPAC 2006 summary talk, S. Suring and his fantastic music band for providing a very comfortable atmosphere during the workshop dinner, N. Okay for his great moderation of a user discussion session, M. Epps for his help during the Jefferson Lab tour, and G. Adams for his beautiful conference photos. We thank Jefferson Lab's Associate Director for Accelerators S. Chattopadhyay for his support of the PCaPAC over two years and Jefferson Lab's Chief Operating Officer M. Dallas for his help in the organization of the workshop.

We gratefully acknowledge grants from the MathWorks Inc. and the Computer Sciences Corporation, which allowed us to provide financial support for PCaPAC participants from Russia and India as well as for students from local Colleges.

Finally, we thank those who participated at the workshop and contributed papers.

The printed version of the PCaPAC 2006 Proceedings is published at Jefferson Lab according to the decision of the PCaPAC International Program Committee of October 26, 2006.

The next PCaPAC workshop will be held in Slovenia or Italy in the fall of 2008.

M. Bickley, P. Chevtsov

PCaPAC International Program Committee

Reinhard Bacher, DESY, Germany
Matthew Bickley, Jefferson Lab, USA
Luciano Catani, INFN, Italy
Ron Chestnut, SLAC, USA
Pavel Chevtsov, Jefferson Lab, USA
Philip Duval, DESY, Germany
Tadahiko Katoh, KEK, Japan
In Soo Koo, POSTECH, Korea
Ajith Kumar, NSC, India
Shin-ichi Kurokawa, Japan
Alexander Lukyantsev, IHEP, Russia
Hiroshi Nishimura, LBL, USA
Mark Plesko, JSI, Slovenia
Deborah Quock, ANL, USA
BeiBei Shao, Tsinghua University, China
Ryotara Tanaka, Spring8, Japan
Ernest Williams, SNS, USA

PCaPAC 2006 Local Organizing Committee

Matthew Bickley - Chair
Pavel Chevtsov
Andrew Hutton
Karen White
Theo Larrieu
Cela Callaghan
Diane Sarrazin

PCaPAC 2006 Schedule

Monday, October 23

5:30 - 7:30	Evening Registration & Welcome Reception

Tuesday, October 24

8:00 - 9:00	Morning Registration
9:00 - 9:30	Opening Ceremony
9:30 - 10:00	"TANGO control system status" J.-M. Chaize (ESRF, France)
10:00 - 10:30	"The new control system for future low-emittance light source PETRA 3 at DESY: from conceptual design work to realization" R. Bacher (DESY, Germany)
10:30 - 11:00	**Coffee Break**
11:00 - 11:20	"Status of the SCSS Prototype Accelerator and Control System" T. Ohata (JASRI/Spring8, Japan)
11:20 - 11:40	"Migrating the STAR Slow Controls System to PCs" J. Fujita (Creighton Univ, USA)
11:40 - 12:00	"Using the common device interface in TINE" P. Duval (DESY, Germany)
12:00 - 1:30	**Lunch Break**
1:30 - 1:50	"The interconnection of TINE and STARS" T. Kosuge (KEK, Japan)
1:50 - 2:10	"Embedding a TANGO device into a digital BPM" G. Gaio (ELETTRA, Italy)
2:10 - 2:30	"Control system for the FFAG complex in KURRI" M. Tanigaki (Kyoto Univ., Japan)
2:30 - 2:50	"Operational experience with synchrotron light interferometers for CEBAF experimental beam lines" P.Chevtsov (JLab, USA)
2:50 - 3:30	**Coffee Break**
3:30 - 3:50	"Beyond PCs: Accelerator Controls on Programmable Logic" M. Plesko (Cosylab, Slovenia)

3:50 - 4:10	"Mono for cross-platform control system environment" H. Nishimura (LBNL, USA)
4:10 - 4:30	"A configurable Interlock system for RF stations at XFEL" M. Penno (DESY, Germany)
4:30 - 4:50	"PC-based magnetic field mapping for superconducting cyclotron (SCC) at VECC" S. Pal (VECC, India)
4:50 - 5:10	"MCS-8 eight axis embedded motion control system" R.Gajsek (Cosylab, Slovenia)
5:10 - 5:30	"Ethernet based embedded system for FEL diagnostics and controls" J.Yan (JLab, USA)
5:30 - 7:00	TINE meeting

Wednesday, October 25

9:00 - 9:20	"Virtual machines for EPICS softioc management" G.Lawson (SNS, USA)
9:20 - 9:40	"MicroIOC: PC and control system longevity" A.Podborsek (Cosylab, Slovenia)
9:40 - 10:00	"Beam line control at EMBL Hamburg" U. Ristau (EMBL, Germany)
10:00 - 10:20	"DAQ based high level software applications using MATLAB" R. Kammering (DESY, Germany)
10:20 - 11:00	**Coffee Break**
11:00 - 11:20	"Ethernet based fieldbus functionality for neutron scattering experiments with PROFINET IO" H. Kleines (FZ Juelich, Germany)
11:20 - 12:00	"XML Tutorial" D. Quock (ANL, USA)
12:00 - 1:30	**Lunch Break**
1:30 - 1:50	"EPICS SCA Clients on .NET x64" H. Nishimura (LBNL, USA)
1:50 - 2:30	"The Project Management Tutorial" M. Plesko (Cosylab, Slovenia)

"Posters in Pills" Session

2:30 - 2:35	"Standardization of the PSI accelerator control systems" A.Mezger (PSI, Switzerland)

2:35 - 2:40	"Status of ISAC Control System" Ch. Payne (TRIUMF, Canada)
2:40 - 2:45	"First operation with SPARC control system" G. Di Pirro (INFN, Italy)
2:45 - 2:50	"EPICS ArchiveViewer Project Status" S. Chevtsov (SLAC, USA)
2:50 - 2:55	"Device Address Redirection as a Tool in the TINE Control System" S. Herb (DESY, Germany)
2:55 - 3:00	"Web GUIs for the TANGO control system" L. Zambon (ELETTRA, Italy)
3:00 - 3:05	"A Communication Protocol for a Distributed Control System with LabVIEW" L. Catani (INFN, Italy)
3:05 - 3:20	**Coffee Break**
3:20 - 5:00	Poster Session
6:00 - 7:00	Networking Reception and Mariner's Museum visit
7:00 - 8:00	Conference Dinner and talk by Dr. Fred Dylla, JLab's Chief Technology Officer and Free Electron Laser project Program Manager "From a Spark in Vacuum to Sparking the Vacuum"

Thursday, October 26

9:00 - 9:20	"User requirements for the PETRA3 control system at DESY" M. Bieler (DESY, Germany)
9:20 - 9:40	"Off-line analysis goes on-line! " M. Lomperski (DESY, Germany)
9:40 - 10:00	"Control systems design a user's perspective" I.Carlino (JLab, USA)
10:00 - 10:30	USER'S PERPECTIVES - DISCUSSION
10:30 - 11:00	**Coffee Break**
11:00 - 11:20	"A prototype of a beam steering assistant tool for accelerator operations" P. Chevtsov (JLab, USA)
11:20 - 11:40	"Application of interest: a relational database approach to managing control system software applications" D. Quock (ANL, USA)

11:40 - 12:00	"Accelerator management with web-based GIS" A. Yamashita, T. Ohata (Spring-8, Japan)
12:00 - 1:30	**Lunch Break**
1:30 - 1:50	"GIS at Jefferson Lab" T. Larrieu (JLab, USA)
1:50 - 2:20	"Status of the CEBAF control system at JLAB" M. Bickley (JLab, USA)
2:20 - 3:00	"Supercomputing research at Jefferson Lab" C. Watson (JLab, USA)
3:00 - 3:30	**Coffee Break**
3:30 - 5:00	Jefferson Lab Tour
5:30 - 7:30	IPC Meeting, CEBAF Center, L102/L104

Friday, October 27

9:00 - 9:30	"Control system studio" M. Clausen (DESY, Germany)
9:30 - 10:30	“Living on the Web” D. Neal (Computer Sciences Corporation)
10:30 - 11:00	**Coffee Break**
11:00 - 12:00	Summary, Isamu Abe Prize, Closing M. Bickley, L. Catani, M. Plesko, P. Chevtsov

Participants of the 6-th PCaPAC International Workshop at Jefferson Lab's CEBAF Center (photo Greg Adams).

The PCaPAC 2006 Isamu Abe prize winners Sergei Chevtsov, on the left, and Jianxun Yan, on the right, with workshop chairman Matthew Bickley, center (photo Greg Adams).

CONTRIBUTED PAPERS

TANGO CONTROL SYSTEM STATUS

J. M. Chaize on behalf of the TANGO teams of ESRF[1], SOLEIL[2], ELETTRA[3] and ALBA[4]

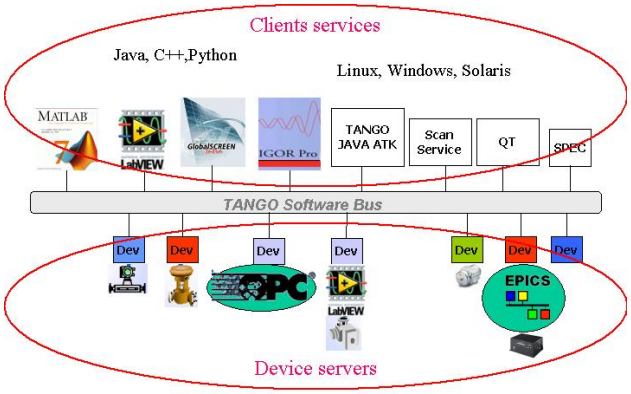
Abstract

TANGO, is a control system framework based on CORBA, it provides distributed objects and distributed services. It can be scaled from controlling a large accelerator complex or a beam line until a small embedded system.

For several years TANGO is being developed as an open source project on Sourceforge by 4 European light sources - ESRF and Soleil in France, Elettra in Italy and Alba in Spain. Over the last year improvements have been made to the TANGO core, web interface, code generator tool and numerous classes has been developed within the 4 core institutes and by other groups around the world. The main highlight of the last year has been the successful commissioning of Soleil. Other highlights include the increasing number of groups using TANGO and the first embedded TANGO device servers now running on a Libera beam position monitor. This talk will briefly present TANGO and then provide an overview of the new developments in TANGO since the last PCaPAC meeting.

HOW IT WORKS

TANGO [1] is an object oriented distributed control system using CORBA developed in a collaborative open source organization by 4 European light sources. It defines a protocol to interact with objects distributed over an heterogeneous network in a so called "software bus".



Much more than a simple software bus, TANGO is now a mature system integrating a large set of tools and utilities necessary to operate a large instrument like an accelerator complex or a simple experiment in a laboratory.

It includes a database server for configuration, a set of tools to administrate and configure it with associated Java GUIs. A History Database system is available to archive all the physical signals of a control system.

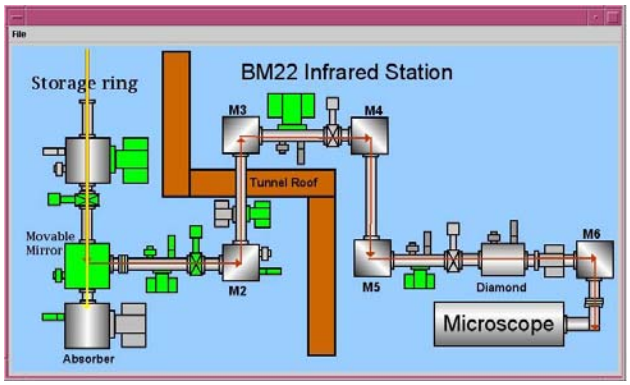
A set of binding allows physicists to work with their favorite tools such as Matlab, Labview, IgorPro or Python to interact with equipment devices.

Many utility classes, servers and their associated GUIs are available to do scanning, alarm system, web interface, etc…

A code generator allows to generate the frame of any server necessary to integrate new hardware in a control system.

Concerning the GUI, a set of generic tools allows to monitor and control all hardware device of the control system.

Specific Java, QT or Python application can be easily written thanks to the corresponding application toolkits. A synopsis edition and animation tool allows to write beautiful synopsis without writing a single line of code.



Recently, two bridges have been written to allow interoperability with EPICS systems. Now, Tango can be used as a object oriented layer above EPICS. In the other way, an EPICS system can also integrate a TANGO server. These 2 bridges open the door to a possible larger integration of these 2 complementary systems.

Another recent progress is the possibility to embed a TANGO server directly in a small acquisition system or in a Field Programmable Gate Array (FPGA). This possibility allows the suppression of one computing layer and open the door of ubiquitous computing for control.

THE COLLABORATION

The collaboration started in 2002 and extended in 2004 and 2005 has shown its efficiency. The 4 collaborating institutes are sharing a CVS repository at Sourceforge[2]. ESRF and ELETTRA are in operation for more than 10 years and use TANGO for their modernization program. At the opposite, SOLEIL and ALBA are both in

*chaize@esrf.fr

[1] http://www.esrf.fr
[2] http://www.synchrotron-soleil.fr
[3] http://www.elettra.trieste.fr
[4] http://www.cells.es

construction phase and use TANGO in a coherent way for Beam lines and accelerator. The collaboration covers a large range of needs.

The responsibility of each institute has been attributed naturally depending of the needs

ESRF, is in charge of the following items:

- Core libraries in C++ and Java.
- Class generator (pogo)
- Database server and browser (Jive)
- Administration tool (Astor)
- Java application toolkit (ATK)
- Synopsis editor (Jdraw)
- Windows setup program.

SOLEIL is in charge of:

- Scada interface
- History database
- Java panels
- Industrial I/O classes
- Many utility classes
- Matlab and labview bindings
- Logging system

ELETTRA is in charge of:

- Database clustering
- Web interface
- Alarm system
- Qt/C++ toolkit (Qtango)
- Porting servers on ARM processor
- Archive events for History Database

And ALBA is in charge of:

- Python server
- Industrial I/O Abstract classes
- Motor control classes
- Motor/Counter device pool
- New web site

The successes

In several domains, several institutes are working together and bring their knowledge and experience.

A typical example is the integration of the Digital BPM Libera [3].

SOLEIL has written the device server for Linux. On the proposal of ELETTRA, ESRF has ported the TANGO libraries for ARM processor. ALBA has written a management tool. ELETTRA has embedded the server in the controller with active help of SOLEIL and ESRF. Now, the four institutes are using the same digital bpm software chain.

Some other narrow collaborative developments can be highlighted:

- Database server clustering and redundancy
- History Database
- Application toolkit and GUIs
- And many other cases.

In all these common field the collaborative development in an open source strategy is a success and TANGO can now be considered as a complete and operational system.

What can be improved

However, there are some fields where things can be improved.

A typical case is the device servers development. With the time, a huge number of hardware has been interfaced and numerous devices servers have been written in all the institutes and by various TANGO users in other labs. (e.g. CEA [4]).

It becomes more and more difficult to benefit from this large basis because there is no easy way to identify, find and classify what is existing and if it may fit to your need. Some database, search machine or classification needs to be setup.

The use of the abstract interfaces defined and agreed by everybody is not used enough. It could largely improve the coherence of the different hardware interfaces.

Concerning the visibility of TANGO. Each institute has its own web site. It does not ease the overview of what's happening. We have started to implement a common web site [5]. The filling of it is in progress.

Lastly, there is a large number of different tools which needs to be learn and known. The integration of all the tools in an unique workbench such as Eclipse[6] would be a good way to have an unified interface.

ENLARGING THE COLLABORATION

Thanks to the download facility, TANGO is now used outside the 4 collaborating institutes. TANGO is used in some schools and universities and in some other laboratories.

We aim to enlarge the community because we thing that more user we are, more rapidly the system will grow and be stable.

The last collaboration meeting held in Grenoble in September has been organized jointly with the ISAC meeting. 11 institutes and 35 people attended to the meeting and shown their interest to TANGO. We discussed the vision of the future and we noticed a common view among the participants.

We agreed to notice that such a big meeting was not the best place to take technical decisions. We would like to continue innovative development without being slowed by too much inertia. However a control is necessary to avoid possible forking or divergence between a large number of collaborative institutes. In consequence we agreed on new rules for decision making.

We setup a management board composed of 4 people. Each institute nominates one member. This management board is in charge of taking decisions concerning the general strategy and the core libraries. They communicate regularly by cyber meeting and maintain together the tasklist. In case of divergence, a vote will be done.

In addition of that, we have defined a certain number of working groups working on particular items.

- History database
- Java Application toolkit
- Industrial I/O servers
- Web site
- Embedded systems
- Etc…

Working group can be setup on a temporary basis for a common development as it has been the case recently for the digital BPM software.

Each working group is taking care of its domain and regularly communicate by cyber meetings. They are all composed by members of different institutes.

Plenary meetings will be organized twice a year in a rotating manner in each institute. It will be the occasion for each working group to report on their activities.

FUTURE DEVELOPMENTS

Even if today TANGO can be considered as a mature system and is stable, active development continues in each institutes to improve and extend its features. The key subjects are the following:

- Merge the 4 web sites in one single http://tango-controls.org. This site has already been created it will be filled up soon.
- Split the documentation in several books and write down more tutorials and code examples.
- Write a distributed naming service in order to decrease load and dependency on database server.
- Complete all the present tools with new features, allow the code generator to generate Python servers.
- Writing numerous abstract classes for widely used devices such as CCD, Industrial I/O, motors ...
- Associate each abstract class with a Java panel.
- Improve the identification, the finding and the packaging of the device server classes available by reinforcing the standardization of the documentation.
- Embedding a TANGO server in FPGAs.
- Work on an Eclipse workbench to integrate all the development and deployment tools in one single interface.
- Last but not least, there is a permanent work of developing new device servers for interfacing new hardware and extend the catalog of supported devices.

ACHIEVEMENTS

Soleil is running

The main achievement of this last year is the successful commissioning of the SOLEIL light source [7]. It is the first light source where both the accelerator complex and the 6 first beam lines are fully controlled by TANGO. This represents 7 instances of TANGO today. 12 in medium term.

The accelerator control system is running about 150 TANGO device servers exporting more than 8000 devices running on 105 hosts (mainly Windows based Compact PCI crates). The Graphical User interfaces are made with a Java based SCADA including ATK widgets. A large part of the control room GUIs has been easily developed by the operator staff themselves thanks to the high modularity of the provided tools.

From the early beginning of the commissioning, TANGO has been providing to the physicists a full range of utility (GUIs, archiving tools, online display of signals etc…) allowing them working efficiently. On the beamline side, the TANGO scan server and associated GUI together with a sequencer named Passerelle allows the users to pilot their experiment.

ESRF and ELETTRA: A careful migration

At ESRF and ELETTRA, TANGO has been step by step and smoothly integrated to the former control system while it was operating 24 hours a day. A set of gateways allows the interoperability of the TANGO control system with the former RPC based control system which were built on the same distributed paradigm.

ESRF achievements and projects

This last year, the Machine control systems runs 350 TANGO servers for 1500 devices while the former RPC system is still controlling 6000 devices. The entire vacuum system, the linac, the interlock system, the alarm system and the PSS have been fully converted to TANGO. Many other instruments are now progressively interfaced to TANGO.

The control room runs a mix of old Motif RPC and Java/TANGO applications. The interoperability is such that any new refurbishment done with TANGO integrates easily to the all.

The beamline are step by step converted to accept TANGO. Today half of them are running some TANGO servers. Servers for key equipments such as CCD detectors, stepper motors or counters are being developed. They will migrate soon on a large number of beamlines.

ELETTRA achievements and projects

At Elettra, several fundamental installations (injector pulsed magnets, RF master oscillator, most of the BPMs) are now fully under Tango control and 140 device servers are currently in operation.

The Global Orbit Feedback project [8] currently under development is completely Tango based.

Elettra is developing two important new projects: the new booster injector [9], which is going to replace the current linac based injector, and FERMI@Elettra [10], a single-pass free electron laser covering the 100 nm to 10 nm wavelength region and capable of delivering femto-second light pulses.

The controls of both these new accelerators will be completely based on Tango.

ALBA status

The civil works have started and the commissioning of the storage ring is foreseen for 2009. TANGO will control both the accelerator complex and the 20 beamlines. For one year, Alba has been participating actively to the collaboration and is developing multiple device servers for a set of industrial I/O. ALBA takes in charge Python server library and a key project named “device pool” This software allows to dynamically instantiate TANGO objects for motors and counters and does the link with dedicated hardware. This project may have a large impact for the beamline control of all the 4 institutes.

References

[1] J.M. Chaize and all. “Tango Control System Framework” PCaPAC2005 Hayama, Japan

[2] Sourceforge project for Tango, http://sourceforge.net/projects/tango-cs

[3] http://www.i-tech.si/products-libera.html

[4] http://www-drecam.cea.fr/scm/lions/python-tango-lions.php

[5] http://tango-controls.org

[6] http://www.eclipse.org

[7] A. Buteau and all “Status of Soleil control systems” EPAC 2006, Edimburgh June 2006

[8] D. Bulfone et al. "Design of a Fast Global Orbit Feedback System for the Elettra Storage Ring", ICALEPCS 2005, Geneva, October 2005.

[9] L. Battistello et al. "The Control System of the Elettra Booster Injector", ICALEPCS 2005, Geneva, October 2005.

[10] C.J. Bocchetta et al. "FERMI@Elettra: A Seeded Harmonic Cascade FEL for EUV and Soft X-Rays", FEL 2005, Stanford, August 2005

THE NEW CONTROL SYSTEM FOR THE FUTURE LOW-EMITTANCE LIGHT SOURCE PETRA 3 AT DESY: FROM CONCEPTUAL DESIGN WORK TO REALIZATION

Reinhard Bacher, Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

Abstract

At DESY, the existing high-energy physics booster synchrotron PETRA 2 will be transformed into a 3rd-generation light source (PETRA 3). In addition, the technical systems and components of the pre-accelerators LINAC 2 and DESY 2 will be improved. Within the scope of this project, the control system and the front-end electronics will be upgraded. Key elements of the conceptual design and the current project status will be presented.

INTRODUCTION

For more than two years, DESY has been changing its scientific profile from a predominantly high-energy physics laboratory to a unique synchrotron light research centre. This change has been manifest in several decisions, namely (1) to switch off the proton-lepton collider HERA 2 and to transform its booster PETRA 2 into a synchrotron light source (PETRA 3), (2) to upgrade the former Tesla Test Facility (TTF) into a user facility (FLASH), and (3) to participate in the European project of a linear-accelerator-driven hard X-ray free electron laser (XFEL) located at the DESY site. In addition to these facilities, DESY also operates the 2nd-generation synchrotron light source DORIS 3.

The future facility PETRA 3 will be a high-brilliance 3rd-generation light source [1]. The design values for the new storage ring will be 6 GeV for the particle energy and 100 mA for the current. The transverse particle beam emittance is expected to be 1 nmrad. Thirteen undulator beam lines operated by HASYLAB (Hamburger Synchrotronstrahlungslabor) and EMBL (European Molecular Biology Laboratory) will provide photons for various experiments (X-ray diffraction and imaging, high-energy resolution spectroscopy, material science, X-ray absorption and resonant scattering as well as structural biology).

At the end of July 2007, operation of HERA 2 will be terminated and all other proton facilities at DESY will be shut down. In the following year, PETRA will be upgraded while the electron or positron pre-accelerators LINAC 2 and DESY 2 continue supplying DORIS III with particles. In order to improve the technical systems and components of LINAC 2 and DESY 2 the pre-accelerators will interrupt service for 6 months beginning 2008, as the re-commissioning of LINAC II and DESY II is scheduled for summer 2008 and the initial commissioning of PETRA 3 for autumn 2008. User beam operation is expected to start in January 2009.

Within the scope of the PETRA 3 project, the control systems and the front-end electronics of LINAC 2, DESY 2 and PETRA will be upgraded almost simultaneously. Therefore, a reasonable balance between continued use of proven concepts or technologies and upgrades using new technologies and ideas has to be found.

FRONT-END ELECTRONICS AND DEVICE INTERFACES

A commonly used front-end hardware standard is mandatory to ensure efficient long-term maintenance of hardware and software. At DESY, hardware standardization was enforced by the DESY-proprietary fieldbus standard SEDAC with decreasing success during the last years. SEDAC is now more than 25 years old and is experiencing strong competition from modern industrial fieldbuses or data links. Besides SEDAC, CANopen is progressively used. Less frequently used are USB, RS232, GPIB, PCI/cPCI and TCP/IP. Increasingly popular is TwinCAT ADS [2] that is also widely used in automation industry.

In order to handle this variety of different solutions within the PETRA 3 project a few measures have been taken:

(1) Only a limited fraction of the existing SEDAC-based electronics will be used in future.

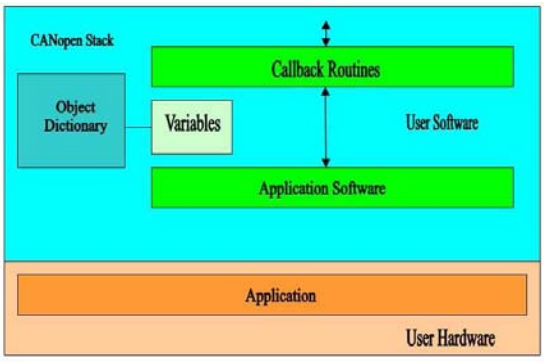


Figure 1: Schematics of CANopen interface software

(2) New developments concentrate on CANopen as fieldbus interface standard. A new standard based on the industrial 3U-Euro crate standard has been established. General electronic boards based on the Coldfire and HCS12 microcontroller [3] families have been developed and the corresponding Vector [4] CANopen implementations have been adapted to our needs [5]. In

order to communicate with the CANopen stack the developer of the application software registers the user specific variables in the CANopen object dictionary and provides the user specific code for a predefined set of callback functions (Fig. 1).

In addition, a processor board based on Altera NIOS II is under development. The general processor boards are connected with user-specific boards implementing the corresponding electrical and mechanical interfaces. The cables to the user-specific equipment are connected at the back of the crate.

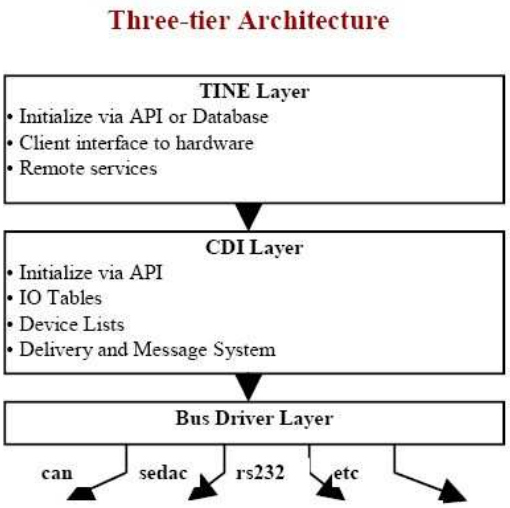


Figure 2: Architecture of the common device interface

(3) A CANopen-to-SEDAC adapter has been developed which allows operating SEDAC-based electronics in a CANopen-dominated environment.

(4) Recently, a common device interface [6] has been developed. It runs as a separate TINE equipment module and offers a generic access to the attached front-end electronics (Fig. 2). Bus plugs are available for CANopen, SEDAC and RS2232. Plugs for GPIB and TwinCAT ADS will be provided.

(5) Specific stand-alone and off-the-shelf test and measurement instruments such as oscilloscopes or spectrum analysers do not in general fit seamlessly into control system architectures. Proper instrument integration often imposes an undue burden on the application developer. To simplify this task, we use IVI-foundation [7] compliant instrument drivers. The Interchangeable Virtual Instrument (IVI) standard defines types of instruments and interfaces to generic virtual instrument drivers in order to avoid vendor-specific incompatibilities. In addition, the VISA (Virtual Instrument Software Architecture) [8] standard is used to become data bus independent. Prototype-like LabView virtual instrument applications for oscilloscopes (Fig 3) and digital multimeters have been finished and tested with instruments from various vendors.

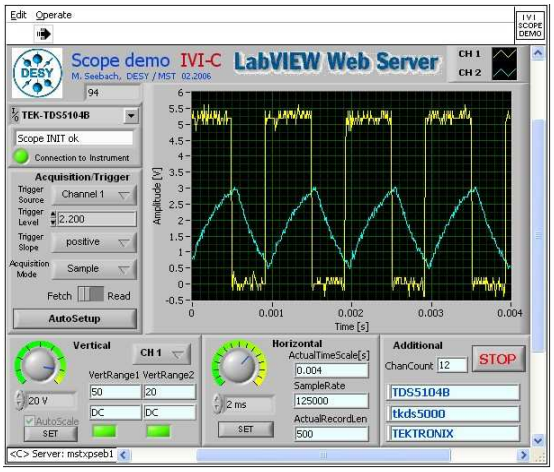


Figure 3: Generic virtual oscilloscope application

TINE CONTROL SYSTEM SOFTWARE SUITE

The up-graded control system uses a multi-layer architecture linked by the integrating middleware or software bus TINE (Threefold Integrated Network Environment), a set of communication protocols and services developed over the past years as the core of the HERA 1/2 control system [9]. TINE is now in a mature state. Figure 4 illustrates the control system architecture.

TINE is a multi-platform system, running on such legacy systems as MS-DOS, Win16, and Vax VMS as well as Win32, Linux, most Unix machines, MACOS, VxWorks and NIOS. TINE is also a multi-protocol system to the extent that IP and IPX are both supported as data exchange protocols. Finally, TINE is a multi-control system architecture system, allowing client-server, publisher-subscriber, broadcast and multicast data exchange.

TINE provides application programmer interfaces (APIs) for Java, VisualBasic, C/C++, LabView, MatLab and a command line interface for scripting tools.

The TINE client/server implementation in C is widely used while the corresponding JAVA implementation has been recently finished.

Name services are provided with plug-and-play automated server registration.

Besides user access control, TINE offers a repeater and redirection mechanisms. The repeater mechanism is suitable for relieving delicate servers such as orbit feedback servers of undue communication burden or data archiving duties. Redirection [10] allows for instance redirecting of locally registered devices (or even 'local' hardware access) to a remote server.

TINE includes interfaces to several associated services. Data filtering and archiving, event handling, alarm filtering and archiving are already supported. An interface for central message processing and archiving will be developed.

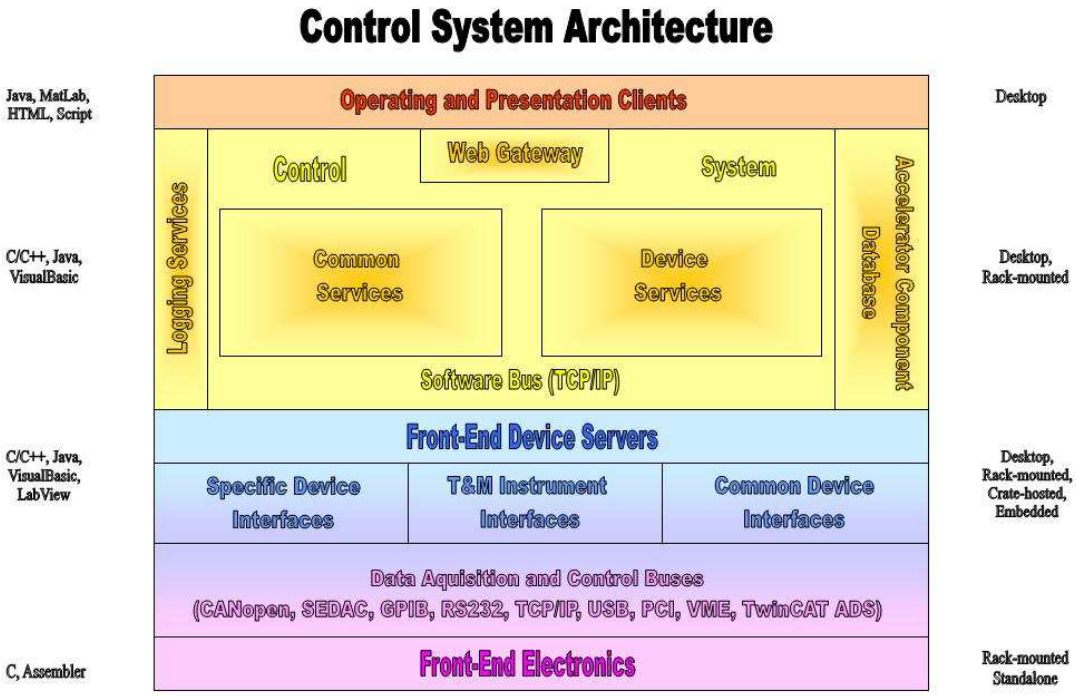


Figure 4: Architecture of the control system for the future light source PETRA 3

A prototype implementation of a read-only TINE Web gateway has been completed which is located in the so-called "Demilitarized Zone" between Internet and Intranet. Data are returned in response to an http-GET request from the corresponding web server.

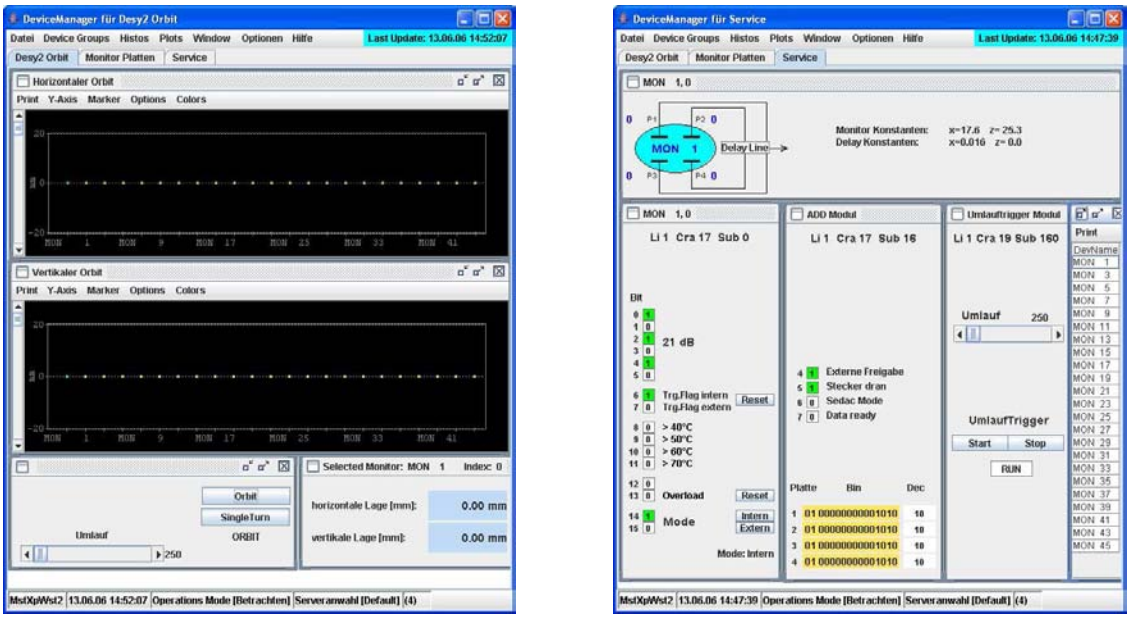


Figure 5: Graphical JAVA client application using both the client application framework and the device manger classes

APPLICATION DEVELOPMENT

To ensure platform independence, JAVA is preferred and recommended as the programming language. Swing lightweight components are the basis for the graphical applications. In addition, The ACOP (Accelerator Component Oriented Programming) toolbox [11] is used for simple data access and rendition. The widely used ACOP chart component is presently being extended to a suite of different ACOP components offering a powerful graphical user interface for rapid development of simple JAVA applications [12].

Framework classes for client and server applications have been developed to ensure design conformity and to handle initialisation data. The so-called device manager, a set of classes dedicated to build standardized operator panels has been provided (Fig. 5). A device server wizard generates code skeletons that are fully operational and integrated into the control system [15]. ANT-based scripts facilitate and standardize application building and deployment [14]

Recently, conceptual design work to implement an accelerator component database as "single-point-of-information" has been started. Fig. 6 shows the proposed architecture.

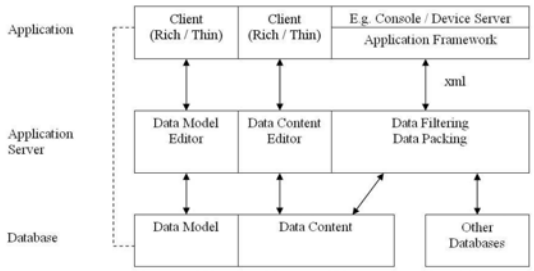


Figure 6: Proposed architecture of the accelerator component database

PROJECT MANAGEMENT

In order to facilitate project management, the Cosy project manager tool from Cosylab [15] has been chosen. This collaborative tool uses extensively e-mail correspondence to distribute task tickets. The corresponding work progress is measured in "minutes worked". Both features may not be applicable in a scientific environment. Most scientific software developers are trained to analyse and structure their tasks without too strict supervision, guidance or control. To promote the acceptance, the tool has been tailored. The progress metrics has been changed from absolute to relative units and the report form has been simplified.

SUMMARY

Within the scope of the PETRA 3 project at DESY, the control systems and the front-end electronics of PETRA 2 and the pre-accelerators LINAC 2 and DESY 2 will be upgraded. PETRA 3 will start user beam operation in January 2009.

Various measures have been taken to limit the variety of device interfaces. In particular, forced use of newly developed CANopen based electronic modules and the use of a common device interface will facilitate this task.

The TINE control system software suite that is now in a mature state, and its associated services will be the core of the upgrades control system of PETRA 3, LINAC 2 and DESY 2.

JAVA is the preferred and recommended programming language. Common class libraries are provided to reduce the final coding work and to ensure design conformity. The ACOP toolbox will be extended to support rapid development of simple JAVA applications even by less experienced programmers.

The Cosylab Project Manger Tool has been adapted for use in a scientific environment. The acceptance by the project team is currently being investigated.

REFERENCES

[1] http://petra3.desy.de/

[2] http://www.beckhoff.de/

[3] http://www.freescale.com/

[4] http://www.vector-informatik.de/

[5] P. Bartkiewicz, P. Duval, S. Herb, DESY, The Common Application Programming Interface for Fieldbus Related Projects at PETRA III, this workshop

[6] P. Duval, H.G. Wu, DESY, Using the Common Device Interface in TINE, this workshop

[7] http://www.ivifoundation.org/

[8] http://www.ni.com/visa/

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

[10] S. Herb, P. Duval, DESY, Device Address Redirection as a Tool in the TINE Control System, this workshop

[11] http://adweb.desy.de/mst/acop/ workshop

[12] H.G. Wu, P. Duval, DESY and M. Plesko, I. Kriznar, Cosylab, The ACOP Family Beans, this workshop

[13] J.Wilgen, P. Duval, DESY, A Device Server Generator for Control Systems

[14] A. Labudda, DESY, Building and Deployment Loosely Coupled Console Applications, this conf.

[15] http://www.cosylab.com

STATUS OF THE SCSS PROTOTYPE ACCELERATOR AND CONTROL SYSTEM

T. Ohata, T. Fukui, T. Hirono, N. Hosoda, T. Masuda, T. Matsushita, T. Ohshima, M. Takeuchi, R. Tanaka, A. Yamashita, JASRI/SPring-8, Hyogo, Japan
M. Kitamura, H. Maesaka, Y. Otake, RIKEN/SPring-8, Hyogo, Japan

Abstract

The prototype accelerator of SPring-8 Compact SASE Source (SCSS) has succeeded in VUV lasing in June 2006. The prototype accelerator was built to confirm the technical feasibility of the 8-GeV X-ray FEL accelerator under construction in SPring-8. A control framework, MADOCA, is used for the prototype accelerator control. MADOCA-based control system worked well and was stable enough to perform VUV lasing operation. This paper described the present status of the SCSS prototype accelerator and its control system.

INTRODUCTION

The X-ray FEL, which is able to produce a high peak brilliance photon beam with high coherency, is required by a wide range of new sciences. The SASE (Self-Amplified Spontaneous Emission) that realizes the FEL light source at X-ray region is the unique concept of the X-ray FEL. Three ongoing X-ray FEL projects exist at SLAC, DESY and SPring-8. In Japan, the X-ray FEL project has been authorized as one of the key technologies of national importance in the 3rd Science and Technology Basic Plan and it is promoted as a national project.

The SCSS project was stated as a RIKEN/JASRI joint project in 2002 [1]. And the project aims generating coherent X-ray beams by SASE with the high gradient C-band accelerator and the short period in-vacuum insertion devices [2]. A 0.06 nm lasing wavelength will be achieved by the project on SPring-8 site at 2010. The SCSS prototype accelerator as the first phase of the SCSS project is the test facility to confirm feasibility. The prototype accelerator, whose electron beam energy is 250MeV and a target photon wavelength is 60nm, was started to build in 2004. The commissioning of the prototype accelerator began in November 2005. On June 20th, the prototype accelerator lased successfully at a wavelength of 49nm with an 110kW maximum output power. Figure 1 shows a picture of the SCSS prototype accelerator seen from electron injector.

Figure 1: Cover shot of the SCSS prototype accelerator. From this side, 500kV Gun tank, 238MHz pre-buncher, 476MHz booster cavity are shown.

A FEL has large differences from traditional SR at the point of coherency. To acquire a high coherent photon beam, a quite stable RF control system and high precision timing system are required. The prototype accelerator has very tight requirements of the RF timing control. Less than a sub-pico-seconds time jitter was required to generate a stable laser. In addition, from a budget problem we should have made ready the control system of the SCSS prototype accelerator by the end of 2005. The X-ray FEL project brought challenges to its control system.

OVERVIEW OF THE SCSS PROTOTYPE ACCELERATOR

One of the important features of the X-ray FEL project is the compactness. A Low-emittance beam injector, a high-gradient C-band accelerator and a short-period in-vacuum undulator are key components and they shorten a total facility length. In the SCSS prototype accelerator we had taken technical adequacy in those key components.

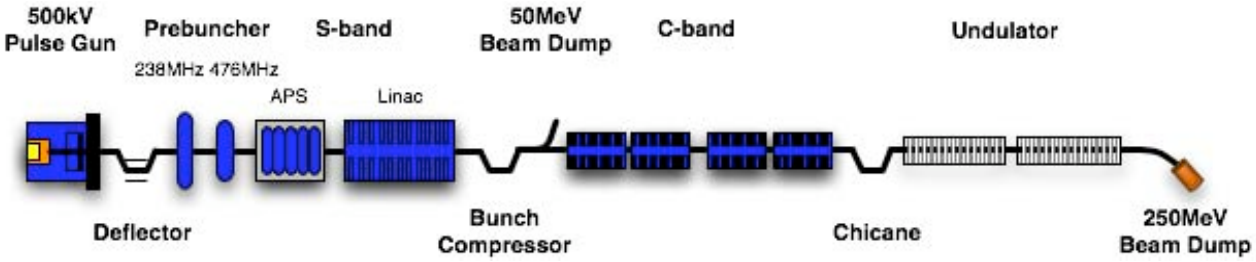


Figure 2: Overview of components of the SCSS prototype accelerator.

Figure 2 shows typical components of the SCSS prototype accelerator.

A 500kV thermionic electron gun generates a low emittance electron beam from CeB6 crystal cathode [3]. The electron injector consists of a 238MHz pre-buncher, a 476MHz booster and an S-band linac as main component.

Two C-band linacs were driven by 50MW klystrons. Their accelerator field comes up to 35MV/m. The beam energy was reached to 250MeV within about 35m accelerating structures.

Two in-vacuum undulators [4] with 15mm period were installed at the end of chicane. In-vacuum design has a merit not only acquiring very short-period magnet array but also providing wider aperture for the electron beam at the commissioning. A 45-deg. tilted magnet configuration has been adopted to ensure wide tenability of correction. The nominal gap width is 3.5mm and its K value is 1.3.

We have confirmed validity of these key technologies by the lasing. Figure 3 shows a plot of generated self-amplified spontaneous emission by the SCSS prototype accelerator. These technologies are promising to make X-ray FEL facility compact.

CONTROL SYSTEM

General configuration

Hardware design of the control system started in April 2004. And we have started development of the software in May 2005. The implementation for commissioning finished in November 2005. Several I/O and communication VMEbus boards were developed for this project. We used MADOCA to shorten the development period of the control software for the SCSS prototype accelerator. MADOCA is a control framework [5] based on database-oriented and network-distributed architecture. An advantage of MADOCA is to abstract a hardware component. We can put easy-to-understand semantic names to all controlled object instead of hard-to-understand syntactic symbols such as bit-number of a DIO board that a VME system has. As a RDBMS, Sybase ASE12.0 is used for management of device configuration data. MADOCA was originally developed for the storage ring control system in SPring-8. MADOCA has scalability because control system applications to X-ray beamline experiments and the accelerator complex (a linac, a booster synchrotron and an 8-GeV storage ring) have been working satisfactory. This feature is promising to develop the prototype accelerator control software at the first stage, developed software components, hereafter, will be applicable to the coming full-scale X-ray FEL facility. Operation program software was developed by using X-Mate; a GUI builder on the bases of X-Window. The GUI builder enables rapid prototyping and easy development of a sophisticated GUI. We should have only written application programs of only GUIs and parts of Equipment Manager (EM). EM is device serving front-end program on the MADOCA framework.

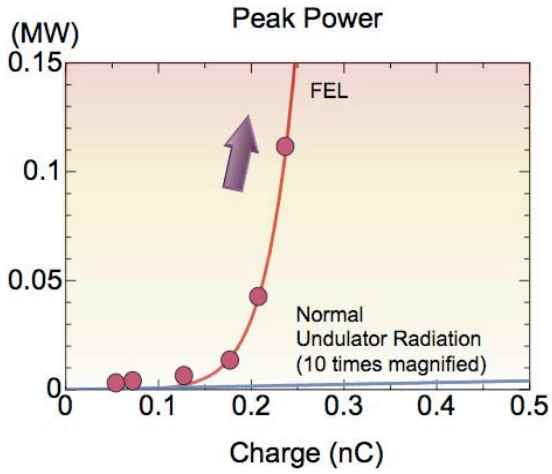


Figure 3: Observed peak powers of first lasing at the SCSS prototype accelerator.

PC-based operator consoles were adopted in the prototype accelerator. We selected Intel architecture (IA-32) workstation running Red Hat Enterprise Linux 3 for the operator consoles. Fourteen VME systems were installed as an equipment interface. As the VME controllers, we used Pentium3 and Pentium-M IA-32 processor boards running Solaris 9 with capability of booting from a Compact Flash card.

The PLC was used as a built-in controller of several components such as a magnet power supply. We selected a FA-M3 from Yokogawa Electric Co., Ltd. The PLCs of main components have graphical display panels with touch pad for local operation. Makers have tested the components by using the display panel of the PLC at own facility. Then we could reduce the task of combined test in site. To reduce the number of signal cabling we adopted the DeviceNet for slow control as a peripheral field bus. The DeviceNet links between the PLC and each accelerator component such as magnet power supply.

We used FL-net for communication between the PLCs and the VME systems. FL-net is one of the Ethernet-based open standard protocols for factory automation (FA) network and was standardized by the requirements of Japan Automobile Manufactures Association (JAMA) [6]. The specification of FL-net was established afterwards as the ISO15745-4 in Oct. 2003, and as the JIS B3521 in Feb. 2004.

A Gigabit Ethernet with optical fiber is used for the backbone of the control network. The network has one subnet of the control networks in SPring-8. The control network including network switch was shared with the FL-net that was separated by virtual LAN. We defined eleven FL-net segments for machine groups such as gun, s-band, c-band and so on.

Figure 4 shows a schematic view of the control system of the SCSS prototype accelerator.

Hardware development

VME boards of high-speed A/D, D/A and two kinds of trigger units were newly developed for the X-ray FEL

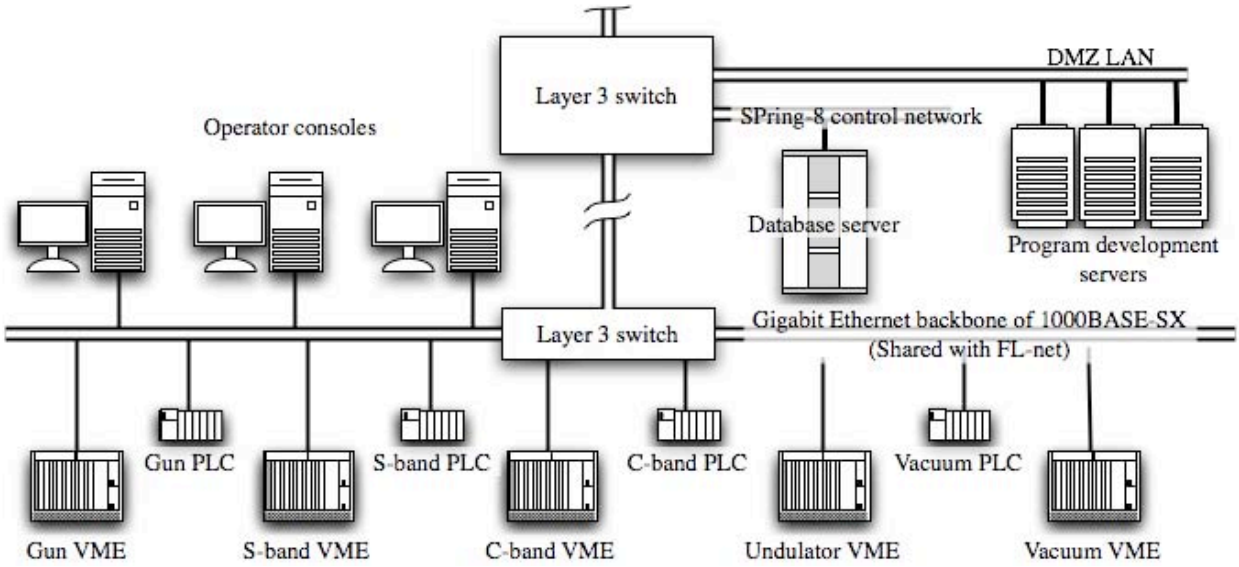


Figure 4: A schematic view of the control system of the SCSS prototype accelerator.

project to improve the stability of the RF system. These boards were compliant with VME specification rev. C.1.

The A/D and the D/A boards [7] control the acceleration gradient variation caused by the instability of the klystron output RF power. The A/D board has four analog input channels, which are running by 238MHz external clock. This clock is the sub-harmonic of C-band acceleration frequency of 5712MHz. For stable analog-to-digital conversion two ADC chips of 12bits resolution are used at each channel. The nonlinearity of maximum likelihood estimation is less than 2 LSB. A Low-pass filter of digital smoothing has a bandwidth of -3dB at 30MHz. Each channel has 4Mbytes memory to accumulate signals from RF-detectors. The D/A board has four analog output channels of 12bits resolution DAC, which are running by 238MHz external clock same as the A/D board. It has 4Mbytes memory and 32 taps FIR filters for each output channel.

We developed a master trigger unit (MTU) and a trigger delay unit (TDU) for the timing system [8]. The X-ray FEL needs a high precision timing system. We selected a tentative target time jitter of 1 pico-second for the SCSS prototype accelerator. The MTU generates a 1-120Hz master trigger to drive all accelerator components synchronously from a commercial AC frequency. The TDU has an oven-controlled 24bits delay counters driven by 238 MHz RF signal and generates delayed pulses resynchronized by C-band acceleration frequency of 5712 MHz. Then we can achieve the targeted value of 1 pico-second. The TDU has independent eight delay modules, which make delayed timing signals for each accelerator components.

A VME-based FL-net interface board has been developed to control devices on PLCs via VMEbus systems. Then, we can control whole equipment of the SCSS prototype accelerator seamlessly from the operator consoles.

We adopted the OPT-VME, and the MCU [9], those stability were proven at SPring-8. The OPT-VME consists of master and slave boards, and is a kind of remote I/O. We used a slave board with D/A function for fast controls of magnet power supply. The MCU is easy customizable intelligent controller based on μiTRON OS with SH-4 CPU. We used the MCU to control stepper motors with position sensors.

For a small amount of data acquisition at many distributed environment, we developed a network-attached equipment with a capability of power over Ethernet (PoE) [10]. A PoE is a technology of supplying power via generic Ethernet cable along with data. Its specifications were standardized by IEEE 802.3af. This technology is useful for low power consumption devices such as a wireless LAN access points. We adopted this technology for a small size data acquisition controller. To realize low power consumption, we developed SH-4 CPU based controller and ported Linux (kernel-2.6.x) OS. Two I/O modules were developed. One is 4 channels Pt100 resistance thermometer module with 0.001 degree resolution and the other is GP-IB module. As a power supply for the controllers, we installed intelligent switching hubs with a capability of power sourcing equipment (PSE) of IEEE 802.3af.

To make easy to write software for network-attached equipment, we developed an application software framework, which called Device Masquerade [11]. Device Masquerade fakes network-attached equipment as local device. Application programmer should not have knowledge of socket API programming. By using Device Masquerade we were able to have incorporated not only MCU and PoE based equipment but also device servers on LabVIEW into MADOCA data collection system.

We introduced GIS-based [12] alarm display for the SCSS prototype accelerator [13]. This is an application program for monitor and notification of machine status. This program was implemented by using ajax technique

[14]. By linkage with the database on MADOCA, we could know troubles on the machine by real-time.

CONCLUSION

8-GeV X-ray FEL project has been authorized as one of the key technologies of national importance and have started construction.

The SCSS control system was developed in 1.5 years, and played an important role for accelerator tuning of VUV lasing. The control system components have proven potentiality to be the building blocks of the 8-GeV X-ray FEL control system. We could have made highly stable RF system and high precision timing system on VMEbus system. Adopting FL-net realized the integration PLC devices into VMEbus system. We could support a small amount of data acquisition of many distributed environment by network-attached equipment. And for these equipment, we developed a application framework; Device Masquerade. and its is working well.

The race for the development of an X-ray FEL has been intensifying worldwide with projects at SLAC, DESY and SPring-8. In Japan, the X-ray FEL has been authorized as one of the key technologies of national importance in the 3rd Science and Technology Basic Plan and it is promoted as one of the national projects. We are starting the detailed design of the control system based on MADOCA for the X-ray FEL.

ACKNOWLEDGEMENT

The authors would like to thank to all contributors in the SCSS project.

REFERENCES

[1] T. Shintake et. al., “X-ray FEL project at SPring-8 Japan”, Proceedings of 8th International Conference of Synchrotron Radiation Instrument (SRI2003), 227 (2004).

[2] T. Shintake, “Status of SPring-8 compact SASE Source FEL project”, NIMA21188, Nuclear Inst. And Methods in Physics Research, A, 507 (2003), 382-397.

[3] K. Togawa et. al., “Emittance measurement on the CeB6 electron gun for the SPring-8 Compact SASE Source”, Proceedings of FEL2004, Trieste, Italy.

[4] T. Tanaka et. al., “Development of the short-period undulator for the X-ray FEL project at SPring-8”, Proceedings of 8th International Conference of Synchrotron Radiation Instrument (SRI2003), 227 (2004).

[5] R. Tanaka et. al., “The first operation of control system at the SPring-8 storage ring”, Proceedings of ICALEPCS’97, Beijing, China, 1997, p1.

[6] http://www.jema-net.or.jp/English/

[7] T. Fukui et. al., ”A development of high-speed A/D and D/A VME boards for a low level RF system of SCSS”, Proceedings of ICALEPCS2005, Geneva, Switzerland.

[8] N. Hosoda et. al., ”Development of 5712MHz synchronous delay VME module”, Proceedings of Particle Accelerator Society of Japan, Saga, Japan, July 2005.

[9] T. Masuda et. al., “Upgrade of the SPring-8 linac control by re-engineering the VME systems for maximizing availability”, Proceedings of ICALEPCS2003, Gyeongju, Korea, 2003.

[10] IEEE802.3af

[11] M. Ishii et. al., “A software framework to control a network-connected equpment as a pseudo device”, Proceedings of ICALEPCS2003, Gyeongju, Korea, 2003.

[12] http://en.wikipedia.org/wiki/Gis

[13] A. Yamashita, et. al., in this conference.

[14] http://en.wikipedia.org/wiki/AJAX

MIGRATING THE STAR SLOW CONTROLS SYSTEM TO PC'S

J. Fujita, Y. Gorbunov, M. Cherney, W. Waggoner, J. Burns, M. Brnicky and R. Thomen
Physics Department, Creighton University, Omaha, NE 68178, U.S.A.

Abstract

The STAR (Solenoidal Tracker At RHIC) experiment located at Brookhaven National Laboratory has been studying relativistic heavy ion collisions since it began operation in the summer of 2000. An EPICS-based hardware controls system monitors the detector's 40000 operating parameters. The system initially used VME processors to communicate with sub-system based sensors over a variety of field busses. The system is being revised transferring user displays from Sun Workstations to PC's and replacing aging VME processors with PC's. The first subsystems were replaced during the Spring and Summer of 2006. The new hardware has been implemented in a way that it can operate in parallel with the existing hardware and software.

I. HISTORICAL BACKGROUND

A. STAR/RHIC description

The Relativistic Heavy Ion Collider (RHIC) at Brookhaven National Laboratory is home to the STAR (Solenoidal Tracker At RHIC) experiment. The STAR experiment investigates collisions between ions accelerated to energies up to 200 GeV per nucleon. In addition, at STAR collisions between polarized proton beams at various energies are also investigated. The primary physics goal of the STAR experiment is to search for evidence of the Quark Gluon Plasma (QGP) [1]. The detectors, detector electronics, and the control system electronics sit in a high radiation area in the Wide Angle Hall (WAH). This area is not accessible during RHIC operation of RHIC Access may be prohibited for days or even weeks at a time. Therefore, the slow controls system as well as the operation of STAR is designed to allow for both remote and continuous operation. Data acquisition electronics are located in the DAQ room adjacent to the STAR control room along with control electronics for monitoring environmental conditions, RHIC clock frequency, and control electronics for interfacing with the accelerator and magnet control systems.

B. STAR Slow Controls system

The STAR controls system utilizes a Sun workstation host in the STAR control room running EPICS (Experimental Physics and Industrial Control System) on a network of front-end processors [3]. STAR slow controls mainly runs EPICS version 3.12.1. Motorola VME board with 680x0 processors running VxWorks version 5.2 are used as the front-end processors in the WAH and DAQ room. These VME cards function as EPICS Input/Output controllers (IOC's). IOC's are connected to the Sun host workstation via a serial connection and an Ethernet connection.

VME boards are configured to boot over the network from the host Sun workstation, since they are only accessible when the RHIC beam is down. Once an IOC is booted, failure of the host will not affect the operation of an IOC unless an IOC is rebooted when the host is down. The use of CANbus provides an alternate method for controlling the electronics in a crate. This design of redundant control capabilities is typical of the slow control design at STAR.

In the control room, the Sun workstation hosts a variety of control software such as EPICS IOC applications for VME boards, the CANbus control application, the MEDM (Motif Editor and Display Manager), the Alarm Handler, and the Channel Archiver. In the control room, each subsystem has at least one workstation on the controls network. These serve as operator interfaces (OPI's) to the control system. During RHIC operation, these OPI's are used for the control of the STAR detector subsystems via their respective MEDM displays. The host Sun workstation display is used to display the Alarm Handler. Through the Alarm Handler many subsystems control displays can be accessed.

II. MOTIVATIONS FOR CHANGE

Current controls system is aging and approaching obsolescence

The STAR control system was designed over ten years ago, and implemented during the commissioning run for STAR in 1999. Since that time, the VxWorks version has never upgraded, and the EPICS version upgraded once from 3.11.9 to 3.12.4. During this time the EPICS has steadily been upgraded and evolved. The current EPICS release, 3.14, became available in 2002.

The outdated version of the EPICS impacts STAR slow controls in numerous ways. It limits the support available from the EPICS collaboration regarding maintaining application tools, and adapting new application tools.

Any further upgrade of the EPICS past 3.13 requires a newer version of VxWorks or alternative operating system. VxWorks license upgrade to meet the near future needs of STAR appears to be costly. Fortunately, the current release has made the EPICS base portable to other operating systems, and IOC's can run on VxWorks, Solaris, GNU/Linux, HP-UX, RTEMS and Win32 [2] This enhancement is a strong motivation for STAR to

migrate to the current EPICS release, and away from VxWorks.

In a similar fashion, VME boards are a decade old. While few have failed, occasionally we have to replace an IOC processor or expand its memory. The detector subsystems currently operating at STAR are constantly evolving. This typically requires adding a second IOC to share load, but often requires additional VME crates with IOC's. In many cases an IOC only accesses the front-end electronics via the serial port. The purchase of additional VME processors appears to be cost ineffective. The ability of an IOC to run on so many architectures under the current EPICS release is again a strong motivation for change.

As with the VME boards, the Sun workstations have proven reliable but are also aging. When the controls system was designed and implemented Sun workstations with Solaris running EPICS was the only option. Today many EPICS users have moved to PC's running Linux and there has been substantial development work done for EPICS on this platform.

All of these reasons for change are becoming increasingly important as new detector subsystems are added to STAR. Under the present controls system environment as new subsystems would come on line and evolve they would be forced to use the costly VME crate MVME solution. They would need to acquire a VxWorks license. New systems were faced with a substantial overhead.

III. NEW SYSTEM

A. Requirements

The upgraded STAR slow control system must meet several requirements. First, the system must be backward compatible with the existing system. Second, the new system must allow for the upgrade of the IOC operating system in a cost efficient manner. The new control system must allow for the migration away from VME boards where possible.

B. Upgrade Plan

While a number of subsystems will keep using VME as IOC processors, the subsystem that does not require the performance of real-time operating system will be replaced by PCs running Linux as host based IOC (soft IOC). These are the subsystems where serial connections are used. By moving these systems to PCs running Linux, the VME processors that are currently in use can now serve as spares for the aging processors. The subsystem that requires the performance of the real-time operating system and VME processors may to migrate to RTEMS. In addition, the host computers in the control room have been migrated to PCs running Linux from Sun workstations running Solaris.

Such a hardware configuration offers several advantages. It is cost effective in that it offers the hope of lower hardware costs for upgrading systems, and lower overhead costs for new systems.

IV. NEW SYSTEM

In the fall 2005, the first Linux PC host computer was installed in the control room along side with Sun Solaris workstations. EPICS 3.13 as well as 3.14 has been installed along with MEDM, Channel Archiver, ALH, and ArchiveViewer. This Linux PC has proven to be very stable great addition for STAR control.

The second Linux PC host computer has just been installed in the control room starting for the run year 2006-2007 as well. The second Linux PC is essentially a mirror of the first Linux PC, functioning as the back up computer.

Starting Spring/Summer 2006, the work on replacing asynchronous device IOC's began. The IOC for the hygrometer that monitors the temperature and relative humidity in the DAQ room has been replaced by Linux soft IOC from MVME 167 board for the run year 2006-2007. The ground integrity detector (GID) in experimental hall is being integrated into EPICS as well. On the VME boards, the work on RTEMS for MVME 167 boards has begun. For the time being, we decided to start with MVME 167 boards. The work on RTEMS for MVME 167 done by at SLAC is also under consideration for MVME 167.

V. SUMMARY

The STAR control system was designed a decade ago and has been fundamentally unchanged since the first experimental run at RHIC in 1999. Both the hardware and software components of the control system are old and approaching obsolescence. The system is at a stage where it is difficult and expensive to upgrade the system. An upgrade of the EPICS base to the most recent version combined with the adoption of Linux and RTEMS as the IOC target architecture will produce many advantages and benefits to the STAR collaboration. A plan is now in place for moving I/O controllers from VxWorks to Linux and RTEMS, transferring user displays from Sun Workstations to Linux PC's, and replacing aging VME processors with PC's.

VI. ACKNOWLEDGMENTS

This work was supported by the Office of Science of the United States Department of Energy.

REFERENCES

[1] 1. J.W. Harris and the STAR Collaboration "The STAR Experiment at the Relativistic Heavy Ion Colider", Nucl. Phys A566, 277c (1994)

[2] R. Cole, M. Kraimer, F. Lenkszus, J. Anderson, J. Hill, et al. The EPICS collaboration, URL: http://www.aps.anl.gov/epics/base/R3-14/index.php

[3] H.S. Matis et al., "Integration and Conventional Systems at STAR", NIM, A499, 802 (2003)

Using the Common Device Interface in TINE

Philip Duval and Honggong Wu, DESY MST, Hamburg, Germany

Abstract

An accelerator control system must in general support a variety of hardware devices and field busses. The Common Device Interface (CDI) is designed to provide the control system engineer with an easy-to-use-and-understand interface for accessing data from the hardware devices, independent from the underlying field bus. The concept behind CDI was initially presented in PCaPAC 2005 [1]. Since that time CDI has undergone numerous refinements which render the writing of a plug-and-play CDI bus plug a straightforward procedure on the one hand, and allow the server developer an intuitive interface for acquiring or setting hardware data either synchronously or asynchronously and from either single or multiple devices.

We note here some of the differences in philosophy between CDI and asynDriver [2] (used by EPICS) and DOOCS [3] device drivers, and we report on the first operational results using CDI on several different TINE [4] platforms (including, Windows, Linux, and Java). Although CDI can be trivially hooked into a TINE server and offers the TINE client interface to the hardware, it is not tightly bound to TINE and could in principle be used independently.

INTRODUCTION

The access and control of hardware devices is typically achieved via fundamental 'Get' and 'Set' operations, where a 'Get' is used to acquire data or status information from the hardware bus and a 'Set' is used to change control modes or download data to the hardware. The details behind these simple operations are in general quite varied for disparate bus types. Some bus drivers offer single-channel read and write calls while others utilize duplex channels for read and write. Some bus drivers are single master, others are multi-master. The bus data format can also be different. For instance, RS232 deals with character string data, whereas SEDAC deals with short integers. Hence the interfaces to these 'Get' and 'Set' operations are generally just as varied as the details behind them.

Prior to CDI, the TINE control system did not have an explicit hardware layer for device servers. For TINE device servers which are EPICS [4] IOCs running Epics2Tine [5] then the device drivers are automatically EPICS drivers (most likely aSyn drivers). For TINE device servers which are DOOCS [6] servers, then the device drivers are DOOCS drivers (which follow the UNIX device server model). By and large the vast majority of TINE device servers at DESY are native TINE servers and follow a "do it yourself" ansatz. Namely, one uses the drivers "which come with the hardware" or (more likely) one uses the in-house SEDAC drivers. This has always proved a viable approach as the bulk of the hardware for HERA and its pre-accelerators is SEDAC. That will change with the advent of PETRA III and is already no longer true with FLASH. There will be considerably more front end hardware using CANOpen devices and TwinCat devices, along with legacy SEDAC, GPIB, rs232, vme, etc.

Application programmers will then either have to become familiar with a bus interface API for each of these bus types or rely on CDI to provide a common interface for all.

The Common Device Interface (CDI) first presented in PCaPAC 2005 [1] has now reached a new level of maturity and is in operation on a couple of test stands at DESY.

CDI API

As all TINE developers are familiar with the TINE client API for accessing data from device servers, CDI strives to leverage this knowledge by offering the same API for accessing data from the hardware bus. In this case a device server running on a Front End Computer (FEC) is a client to its attached hardware.

CDI itself offers a CDI-native API which is TINE-similar. However, by and large developers will want to make use of precisely the same TINE client API calls as used when accessing data from any other end-point in the control system.

It is worth a bit of time to review the TINE client API, as it does not follow the (seemingly ubiquitous) 'get(), set(), and monitor()' APIs so loved by other control systems. Instead TINE deals with 'calls' in the sense of Remote Procedure Calls or Remote Method Invocation, which are passed via data 'links'. A data link can be either synchronous or asynchronous and calls a TINE property at a TINE endpoint. A TINE endpoint is in turn determined by a namespace consisting of device context, device server, and device name. Schematically, a call will attempt to access:

/<context>/<server>/<device> [<property>].

and will optionally send an input data object to the target and/or request an output data object to be returned. A TINE property should rather be thought of as a 'method'. If data is both sent to and received from the target, this exchange of data objects occurs atomically. Note also the requested data access (read or write) is separated from the

data objects. That is, a 'read' call which needs to send data to the target is still a 'read' call!

The synchronous API call is ExecLink() (short for 'execute link') and has the following basic prototype:

ExecLink*(name, property, dout, din, access, timeout)*

The corresponding asynchronous call AttachLink() has additional monitoring parameters which supply callback information as well as the monitoring 'mode', which can contain a wide variety of monitoring instructions.

AttachLink*(name, property, dout, din, access, pollrate, callback, callbackId, mode)*

One can of course wrap these calls with get(), set() and monitor(), but will loose generality in doing so. Indeed some general features are then difficult to re-introduce, such as a set-get atomic operation or starting a monitor with instructions to stop processing until the first update, etc.

Using these API calls to access the hardware to access the local hardware becomes a simple matter if the endpoint uses the device context "localhost", and the device server "cdi". Special parsing of the full device name also allows multiple endpoints with a single call. For instance, a call to "/localhost/cdi/#1" would access only the device registered as device number 1. On the other hand a call to "/localhost/cdi/#1-#100" or "/localhost/cdi/#1,#3-#10,#99" would identify the individual registered devices and access them as a group. The CDI property space include the properties "RECV" for receiving (reading) data from the device, "SEND" for sending (writing) data to the device, "RECV.SEND.ATOM" and "SEND.RECV.ATOM" for issuing a pair-wise read-write or write-read operation which is guaranteed to be atomic, "RECV.CLBR" and "SEND.RECV.CLBR" for returning data which has been calibrated according to the registered calibration rules, and such properties as "ADDR" and "BUSNAME" which return information about the endpoint device.

For the sake of an example, a CDI monitor call in C might look like

```
dout.dArrayLength = 100;
dout.dFormat = CF_UINT16;
dout.data.sptr = rbData;
AttachLink("/localhost/cdi/#1-#100", "RECV.CLBR,
        &dout, NULL, 1000, cb);
```

which would read devices 1 to 100, calibrate the data, fill in the read-back buffer rbData and call the callback routine cb at 1 Hz.

Needless to say, there is a similar interface for other languages such as java, Visual Basic, or LabView. We should also point out that CDI devices are registered both with a device name and a device number. The registered device name can likewise be used in the above calls, which might be desirable when browsing the registered hardware remotely. Generally speaking, a server developer will be more inclined to use the device number when accessing the attached hardware, since the actual name of temperature sensor, sputter pump, BPM or whatever in question is mostly irrelevant at that level of data access.

CDI DETAILS

CDI operates on a plug-and-play basis, and adding a new bus interface plug to CDI only involves writing a new bus interface plug. The CDI shared library needs only to be compiled and installed once for the platform in question. On windows for instance this will be cdi32.dll (or cdi64.dll) and on Unix systems libcdi.so. Application platforms such as java, VB, or LabView will also access this same shared library.

When the library loads, it will look for a CDI bus manifest file, which is a simple comma-separated-value file and can be a simple as a single column with a list of bus plug libraries, as shown below in figure 1.

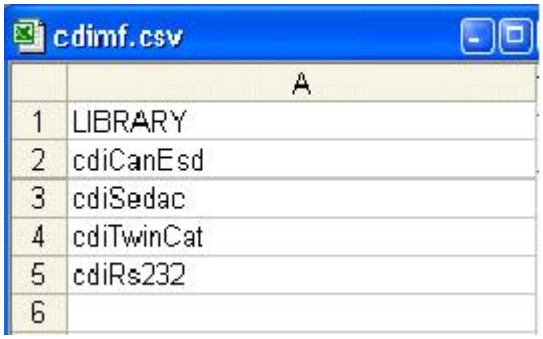


Figure 1. Example of a CDI bus manifest file.

CDI will then call cdiLoadLib() for each bus plug entry in the manifest list, for instance

cdiLoadLib("cdiCanEsd.dll");

on windows or

cdiLoadLib("libcdiCanEsd.so");

on Unix platforms, etc. If the library loads successfully, it will (via its prologue code) register its name and all of its bus handlers with CDI, which itself has no a priori knowledge of any hardware bus interface.

After the manifest has been read, CDI will look for a CDI device database and, if found, read it and register all devices and device information contained within. The registered information will include the bus name and address of the device and any accompanying bus parameters (such as bus speed) along with its assigned device name and number, as well as data access parameters and calibration rules. The supported calibration operations include addition (and subtraction), multiplication, and exponentiation along with bit shifting

and modulo arithmetic on integer values. There is no limit to the number of calibration rules which can be applied, and they can be applied in any order, although care must be taken when mixing possible floating point rules such as multiplication or exponentiation with purely integer rules such as modulo arithmetic or bit shifting, so that the outcome of the calibration makes sense. We note here for completeness that CDI can operate without a database, but then all devices must be registered via API calls from with the server application.

It should also be pointed out that writing a new bus plug for CDI is a relatively straightforward process. CDI itself will do nothing but load the bus plug library. It is the duty of the bus plug to register itself with CDI. There are a handful of CDI routines the bus plug should make use of in order to function properly. These are essentially all registration routines which provide CDI with the bus handlers for accessing its bus hardware (calls to open the bus, read and write to the bus, and close the bus). If a new hardware device has a driver API for the target platform, then the task of writing a bus plug is no more complicated than wrapping the appropriate API calls within the CDI bus plug handlers.

To this end, a "bus" plug does not itself have to interface directly to hardware, but could simply provide access to more complicated, generic hardware entities, such as "stepper motor" or "oscilloscope". These entities could themselves access the hardware directly (using CDI in a nested manner) and incorporate the "business logic" which handles the generic functionality the all "stepper motors" or all "oscilloscopes" have.

REMOTE CDI

TINE servers which make use of CDI will automatically offer the full palette of device access available locally to remote clients. In other words, CDI will export a de facto device server (unless operating in stand-alone mode) which makes the hardware available for remote access. A remote client wishing to access the device hardware directly can do so by contacting the endpoint using the device context of the registered TINE server and the device server name given by the Front End Controller (FEC) name appended with the suffix ".CDI". In other words, a Beam Position Monitor FEC, called BPM which registers itself in context PETRA will automatically export a device server called BPM.CDI also in context PETRA. The local server will access its hardware using, for example, the endpoint

"/localhost/cdi/#1-#100"

whereas a remote client could access the hardware via

"/PETRA/BPM.CDI/#1-#100"

Indeed, remotely it might make more sense to use device names and issue the call as

"/PETRA/BPM.CDI/BPM/OR1 – NL25"

assuming that "OR1" is the registered name for device 1 and "NL25" is the registered name for device 2.

Although these remote services are automatically present and do not require coding, accessing the hardware devices remotely in this manner should be thought of as a debugging service. On some (rare) occasions, when the server itself has no more complicated duties than to read out a number of, say, temperature sensors, calibrate the results and offer them, then the default CDI server can be used almost as is (It would probably be prudent in this case to alias the CDI property "RECV.CLBR" with "Temperature", for example).

CURRENT STATUS

CDI is now being tested at DESY for isolated but relevant cases in HERA and PETRA, and is so far proving to be stable as easy to use as advertised. The existing bus plugs include three varieties of SEDAC bus plugs (which suggests that these should rather be thought of as bus interface plugs) for both Windows and Linux, two varieties of CANOpen for Windows and Linux, RS232 for Windows and Linux and most recently for the TwinCat interface from Beckoff [7] (for Windows).

Current efforts will now focus on providing database management tools which allow rapid and straightforward creation and checking of the CDI database.

REFERENCES

[1] R. Bacher, et. al., "Common Device Access for Accelerator Controls", Proceedings PCaPAC 2005.
[2] M. Kraimer, E. Norum, and M. Rivers, "asynDriver: Asynchronous Driver Support", http://www.aps.anl.gov/epics/modules/soft/asyn.
[3] http://tine.desy.de.
[4] http://www.aps.anl.gov/epics.
[5] Z. Kakucs, P. Duval, M. Clausen, "An EPICS to TINE Translator", ICALEPCS 2001.
[6] http://doocs.desy.de
[7] http://www.beckhoff.com/english

THE INTERCONNECTION OF TINE AND STARS

Takashi Kosuge*, Philip Duval**, Yasuko Nagatani* and Kazuyuki Nigorikawa*
*High Energy Accelerator Research Organization (KEK)
**Deutsches Elektronen-Synchrotron (DESY)

Abstract

We have succeed to connect TINE (Three-fold Integrated Networking Environment)[1] and STARS (Simple Transmission and Retrieval System)[2][3] with co-development of gateway and bridge programs.

TINE is very powerful system which was developed by DESY and STARS is a very simple message transferring system for small scale control system. At the Photon Factory(KEK), STARS is used for synchrotron radiation beamlines and installation of STARS is still in progress.

Recently, we have started development of a ring information (ring current, life time etc.) distribution system for the beamline with TINE and STARS. Multicast function of TINE works efficiently in this system.

We will describe the detail of connection TINE-STARS and development status of ring information distribution system.

OVERVIEW OF TINE

TINE is a significant networking environment and it has advantages as follows.

- Multi-Platform (Windows, Unix, Macintosh, VMS, VxWorks, DOS, etc.)
- Multi-Protocol (IP, IPX)
- Multi-Architecture
- Plug and Play

Client and server components are integrated and behave like a “software bus”.

Fig. 1 shows image of TINE components and their communication.

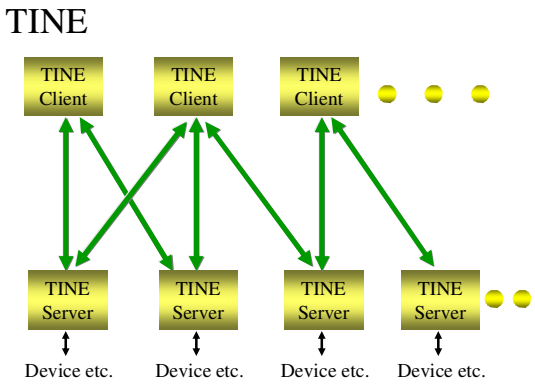


Figure 1: TINE servers and clients.

OVERVIEW OF STARS

STARS is a message transferring software for small-scale control systems with TCP/IP sockets, which works on various types of operating systems (STARS server is written in Perl). STARS consists of client programs (STARS clients) and a server program (STARS server), and each client is connected to the server via a TCP/IP socket. STARS users can upgrade the system by writing client programs, and STARS clients are able to participate in the system at any time without system stoppage.

STARS server and clients handle only text-based messages and these text messages are transferred through TCP/IP socket. This brings extremely simple architecture for STARS.

Hierarchical node name

Each client of STARS has its own unique node name. The node name is used to identify nodes at transferring messages. A hierarchically structured system can be developed with the hierarchical node name of STARS. A period (“.”) is used for the separator and the STARS server uses the first part of the destination. For example, the message which has “Br2.Dev1” as destination will be delivered to “Br2”.

Fig. 2 shows example of STARS connection and transferring messages in STARS.

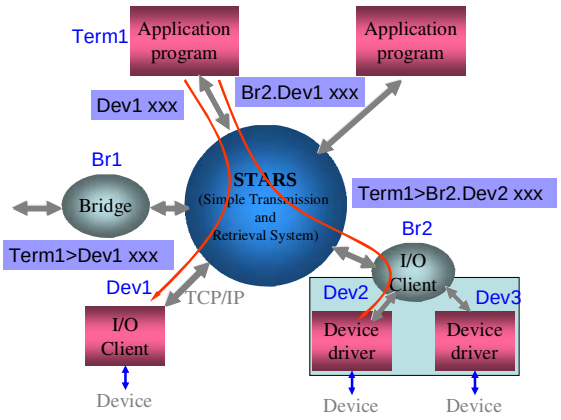


Figure 2: STARS connection and messages.

THE INTERCONECTION

We made 2 types of interface (see Fig. 3). One is to access STARS from TINE clients (TINE-STARS gateway) and the other one is to access TINE servers from STARS (TINE bridge). Development of these interfaces is

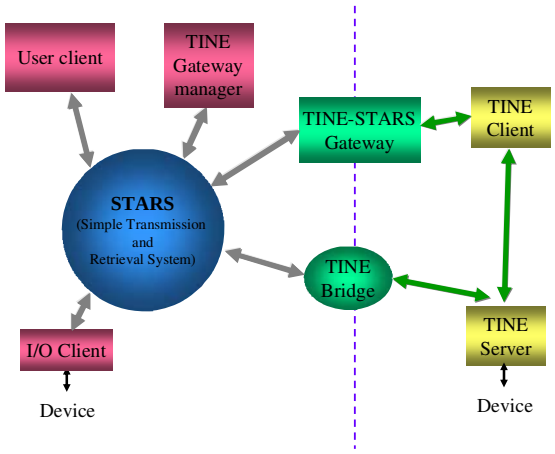


Figure 3: The interconnection TINE and STARS.

still in progress now. We will describe recent version of interfaces here.

Name space

The name space is very important to identify devices etc. in both systems. TINE uses "Context Name" and "Device Server Name" to specify an equipment module on a FEC (Front End Controller), and "Device Name" to identify an individual device managed by the equipment module. Properties (or methods) of the device are accessed via the device "Property". STARS uses a straightforward hierarchical structure as its name space. One of the big issues of interconnection is "How to map the name space".

We mapped TINE device name and property name to the STARS node name by separating identifiers of TINE with the "period" character.

TINE-STARS gateway

TINE-STARS gateway is used to access STARS from TINE clients. The gateway is basically a STARS client and it looks like a TINE server for TINE clients. The gateway exchanges requests from TINE clients and sends

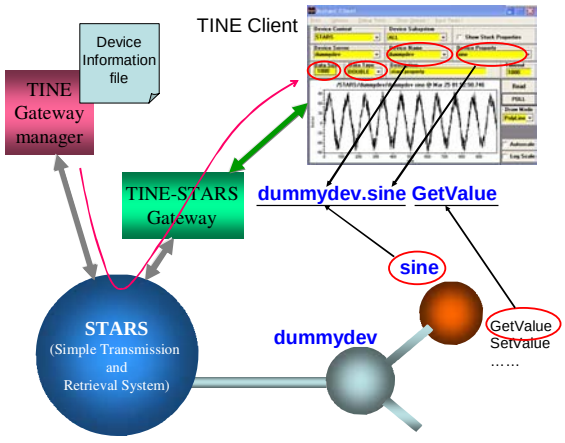


Figure 4: Accessing STARS from TINE client.

or gets corresponding messages from STARS. Then, the TINE-STARS gateway maps the name space between TINE and STARS in both directions. Fig. 4 shows accessing STARS client from TINE and example of mapping STARS name space to TINE.

Gateway manager

TINE can handle various data types such as CF_TEXT (C: char, VB: String, Java: char), CF_LONG (C: long, VB: Long, Java: int), CF_FLOAT(C: float, VB: Single, Java: float) etc. and user-defined data types are available.

On the other hand STARS does not need to process different data types because STARS handles text messages only. Each STARS client must exchange the text into corresponding data type by itself. This means the TINE-STARS gateway has to know how to convert TINE data types to text and how to parse STARS data text into TINE data types. The gateway manager (also STARS client) does just that!

TINE bridge

A TINE bridge is used to access TINE servers from STARS. It exchanges name space and data type between

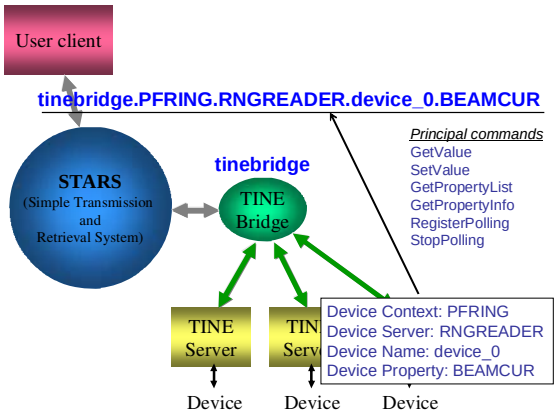


Figure 5: Accessing TINE server from STARS.

TINE and STARS. If the bridge receives "GetValue" or "SetValue" command, the bridge reads or writes corresponding device property of TINE server. Fig. 5 shows example of accessing TINE server from STARS.

The bridge supports attach link and multicast function of TINE then it can get values from TINE server frequently. Device properties must be set in configuration file of TINE bridge at present. Then the bridge sends "_ChangedValue" event message to the STARS server.

APPLICATION

At this time, we have developed a beta version of the PF ring information display with TINE and STARS.

The system consists of a TINE server which gets ring status, TINE bridge and "display client" of STARS. The TINE server is connected to PF ring control system and gets information of ring such as beam current, life time, operation mode, gap of undulater etc. Then it sends them

to the TINE bridge frequently with attach link and multicast.

After receiving value, the bridge checks the value has been changed by comparing with previous value. If value has been changed, the bridge sends "_ChangedValue" message to STARS server.

"display client" sends "GetValue" command to the bridge and get values at starting time. Then it sends "flgon" command to STARS server and waits event message which shows value has been changed.

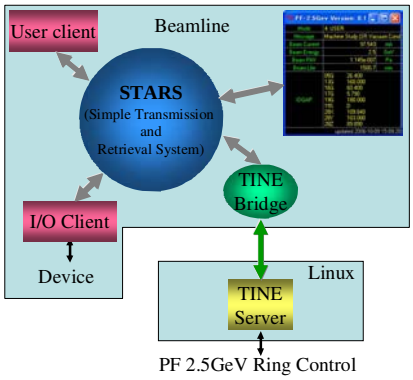


Figure 6: PF ring information display.

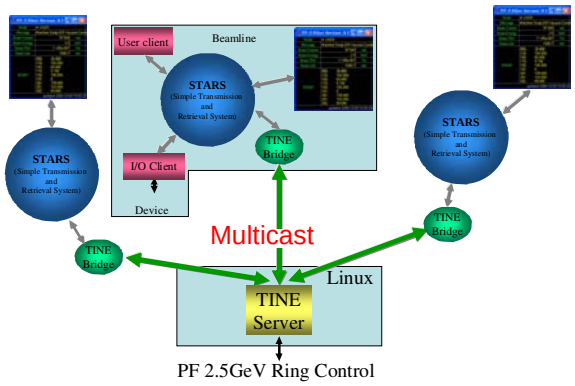


Figure 7: Information display with multicast.

Usually this TINE bridge and "display client" will be attached to a STARS server that works on beamline control system. And, we prepared Linux PC for the TINE server and ENS of TINE. The TINE server which gets ring status is running on this PC and waiting for connection from TINE bridges. Fig. 6 shows composition of the system.

We are planning multiple connections as next step. Number of beamline control system attached TINE bridge will run finally. Multicast function of TINE will work effectively (see Fig. 7).

CONCLUSION

This time we have succeeded to connect TINE and STARS to develop a TINE-STARS gateway, a gateway manager and TINE bridge. This means;

- STARS can connect various systems via tine,
- TINE can connect COACK[4][5][6] etc. via STARS,
- Efficient function on TINE and STARS is available.

At present, there is an unresolved character problem in the name space mapping. TINE allows blanks and other special characters within a device name. STARS, however uses the standard set "A" to "Z", "a" to "z", "0" to "9", as well as "_" and "-". Development is still in progress and the problem will be fixed.

REFERENCES

[1] http://adweb.desy.de/mst/tine/

[2] http://pfwww.kek.jp/stars

[3] T. Kosuge, et al., "Recent Progress of STARS", PCaPAC2005, Hayama, 2005.

[4] I. Abe, et al., "Recent status on COACK project", PCaPAC2000, Hamburg, 2000.

[5] T. Kosuge, et al., "COACK Application for the Beamline Interlock System at the Photon Factory", PCaPAC2000, Hamburg, 2000.

[6] T. Kosuge, et al., "COACK Multi-Server System with STARS", PCaPAC2002, Frascati, 2002.

Embedding a TANGO device into a digital BPM

C. Scafuri, V. Forchì, G. Gaio: Sincrotrone Trieste, Trieste,Italy
N. Leclercq: Synchrotron SOLEIL,Gif-sur-Yvette,France

Abstract

The Global Orbit Feedback project will provide the Elettra storage ring with a fast digital orbit correction system. The goal is to reach sub-micron stability and to damp disturbances up to 150Hz. A fundamental component of the new feedback system are the digital BPM detectors "Libera Electron" by Instrumentation Technologies. The Tango control system has been ported to the Intel Xscale Linux used by the embedded controller of the Libera devices. We have already developed, tested and successfully deployed the embedded version of a Tango Device server on the Libera device. Most of this work has been done in collaboration with the other partners of the Tango collaboration.

GLOBAL ORBIT FEEDBACK

The Global Orbit Feedback project goal is to equip the Elettra storage ring with a fast digital orbit correction system. The main specifications are: sampling rate of 10 kHz, sub-micron resolution, capable of damping disturbances up to 150 Hz and suppress mains generated disturbances up to 300 Hz. A fundamental component of the feedback system is the new digital BPM electronics.

Digital BPM

The new digital BPM detectors are based on Instrumentation Technologies Libera Electron devices. These devices are connected to the existing storage ring pickups, replacing the old 96 analog BPM detectors. They provide the required resolution and bandwidth. The digital processing is done entirely in a FPGA module. The Libera Electron device is also equipped with a single board computer (SBC) based on ARM type processor (Intel Xscale). The SBC runs a custom version of Linux and is in charge of managing the device. The Libera detector provides data at three different rates: turn-by-turn, 10 Hz and 10 kHz. The first two data flows are available to the SBC, while the latter is available only to some custom FPGA interface. Instrumentation Technologies provides a software library, named Control System Programming Interface (CSPI), and a gnu-gcc based cross-compiler development environment to allow the development of custom applications for the digital BPM. These applications can either be deployed on the SBC itself or can talk to the SBC through a dedicated socket server, which exports the CSPI api.

TANGO DEVICE FOR LIBERA

Both Soleil and Elettra control systems are based on Tango [1],[2], and both institutes use Libera as BPM detectors. In order to easily integrate the Libera devices with the rest of the control system, Soleil designed and implemented a Tango device using the CSPI library to get data from the BPM and provide the needed configuration and management capabilities. The first version of the Libera tango device server (DS) accessed the SBC via the socket server. Although this version provided all the required functionality, it had the drawback of a complex deployment scheme and of poor resource usage, for example network bandwidth, which was several times higher. The availability of the full source code of Tango and of the cross development tools for the Libera SBC, led us to develop an embedded Tango Device server for the Libera SBC. In order to do this, three steps had to be performed: porting of omniORB to ARM, porting of Tango to ARM, development of the Libera Device Server with new the tools and the embedded version of CSPI. The advantage of deploying an embedded Tango Device server is twofold: reduction of network traffic and simpler deployment, troubleshooting and maintenance.

omniORB for ARM

The Tango control system is based on omniORB, an open source CORBA implementation [3]. The problem of porting the omniORB to the Libera device was the most difficult.

building in a cross-development environment The first difficulty is due to the fact that we had to build omniORB with a cross compiler set of tools. The building of omniORB happens in three stages. During the first stage the IDL compiler and associated tools is built. In the second stage the IDL compiler is used to generate the parts of the core omniORB library and other parts dealing with basic CORBA services, stating from a set standard IDL files. The third stage compiles the core libraries, the basic stubs and skeletons and generates the effective omniORB library and related tools. The problem of using a cross-compiler arises during the second stage: the new IDL compiler is built for the target architecture (ARM) but it is actually run on the host architecture (i386). This problem is solved by stopping the build process, manually copying the IDL compiler and related libraries from another "native" build tree, which has been already built, and then restarting the build process. A definitive solution would involve a complete restructuring of autoconf/automake based build

process. After completing the third stage, we had to perform yet another manual intervention, patching by hand one of core omniORB libraries. This step, performed by means of the arm-ar tool, comes from the fact that in many cases the linker - arm-ld - is not able to produce a working executable and notifies some unresolved symbols.

coping with endianess The second difficulty derived from the peculiar format of type "double" on the ARM Intel Xscale processor. By default the ARM processor is little-endian (like i386, VAX ...) but can also be programmed to be big-endian (like 68k, HP-PA, the network ...). But on this processor, as supported by the gnu-gcc based cross compiler, the two 32 bit words of a double have the same order as in big-endian processors. This peculiar case of mixed endianess was not originally handled by omniORB, resulting in mangled numbers when data of type double is exchanged with other types of processors. After we identified the problem we were able, thanks mostly to the synergies of the Tango collaboration[4], to get in contact with the original developer of omniORB. He provided the necessary patches, while the debug and testing was carried out by us. After a few iterations we had a perfectly working omniORB for the ARM processor on Linux.

Tango for ARM

Building Tango on ARM was relatively straightforward. The only special step we had to perform was to regenerate the CORBA stubs and skeleton classes with the patched omniORB tools. After this step the standard configure/make/make install procedure was applied. An extensive set of tests have been carried out. For still not completely clear reasons, we had to link programs using Tango and omniORB as fully static executable (that is, avoid shared libraries). By sticking to this simple rules our tests for omniORB and Tango worked correctly.

Embedded Tango Device

Porting the prototype Tango Device for Libera that was working with the socket version of the CSPI library to run natively inside the Libera SBC was extremely simple. After some trivial changes in the Makefile the device was compiled without major modifications. After some tests we made some further changes on the thread handling code due to differences in the CSPI library behaviour. There were no problems or bugs due Tango or omniORB. We deployed the new embedded Tango device for Libera on 96 nodes with any problems. Figure 1 is a picture of the Libera devices installed in one section of the Elettra storage ring. All the already running programs that accessed the Libera devices continued to work without modifications.

RESULTS

By embedding the Libera device we achieved three important results: dramatic cut of the bandwidth required by the Libera BPM traffic; simplify and make more intuitive the deployment and management of new Libera devices; free a lot of wasted processing power on the "intermediate server" and exploit instead the unused processing power of the Libera SBCs. Tango speed performances are limited by the network performances of the Libera SBC for almost all the operations; only operations involving doubles suffer some penalization due to the additional operations required by the weird format of doubles on ARM. Since the BPM data is of type float, this is not a real problem. In practical terms, we get the same results of the non embedded version using the socket based CSPI library, that is about 8ms for reading a single position from one BPM and less than 50ms for reading the whole orbit on both planes. Since Libera is a Tango device, we could exploit all the Tango infrastructure and developing tools to rapidly design and put in operation dedicated control panels for the new BPM (see Figure 2 and Figure 3). The embedded Libera device has been in daily operations at Elettra and Soleil for more than two months. No problems have emerged so far due to omniORB or Tango.

Figure 1: Libera installation for one section of the storage ring

CONCLUSION

The embedded version of the Tango Libera device is now in operation in two institutes: Soleil and Elettra. Its adoption greatly simplified the deployment and management of new Libera BPM in the Elettra storage ring. The procedure became so straightforward that we were able to upgrade the old BPM with Libera during some users shifts, exploiting the few hours that are weekly dedicated to machine tuning and calibration. No service disruptions or

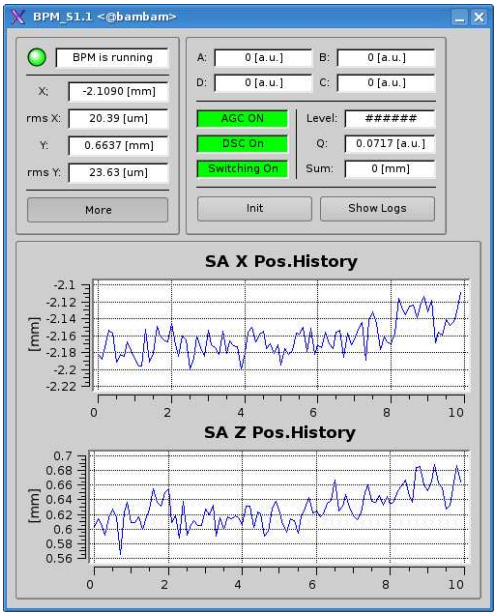


Figure 2: control panel for a Libera BPM

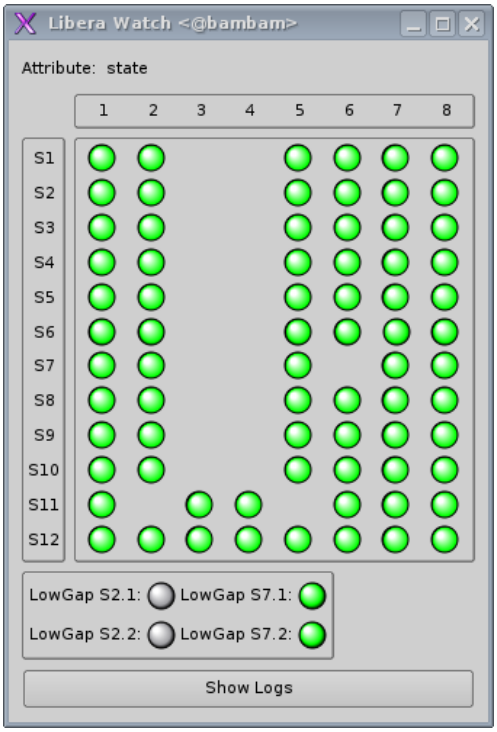


Figure 3: panel for monitoring the status of all the Libera BPMs of the Elettra storage ring

important malfunctions have so far emerged. Performances are limited only by the TCP/IP stack of the Linux version running on the Libera SBC. The porting of Tango to the ARM architecture is a success of the Tango collaboration. It showed once again that Tango is portable and fit to run also on embedded hardware. As a by-product of the effort, we have now a version of omniORB and Tango for the ARM processor. This fact opens up the possibility on using Tango on a series of interesting devices such as embedded instrumentation, PDAs, cellular phones, etc. . .

ACKNOWLEDGEMENTS

Special thanks to Duncan Grisby, who provided the patches for handling the mixed-endian doubles on omniORB.

REFERENCES

[1] A. Götz et al. “Tango a CORBA based Control System”, ICALEPCS 2003, Gyeongju, Korea

[2] J.M. Chaize “Tango control system status”, these proceedings

[3] http://omniorb.sourceforge.net

[4] E. Taurel “The Tango Collaboration Status and some of the Latest Developments”, ICALEPCS 2005, Geneva, Switzerland

Control system for the FFAG complex in KURRI

M. Tanigaki*, K. Takamiya, H. Yoshino, N. Abe, T. Takeshita, Y. Mori, K. Mishima, S. Shiroya
Research Reactor Institute, Kyoto Univ., Kumatori, Osaka 590-0494, Japan
Y. Kijima, M. Ikeda, Mitsubishi Electric Corp. Kobe 652-8555, Japan

Abstract

The 150 MeV proton FFAG accelerator complex in KURRI is now in the final stage of construction. This accelerator complex will be served as the proton driver for for the feasibility study of accelerator driven subcritical reactor (ADSR). The control system for this FFAG accelerator complex is based on conventional PCs and programmable logic controllers (PLC) on TCP/IP network. The databases of parameters for connected devices are maintained by PLCs and these can be accessed by any type of equipments or higher integrated applications as long as they can handle the conventional network connection. We report the current status of the controlling system and future upgrades for the beam commissioning of our FFAG complex.

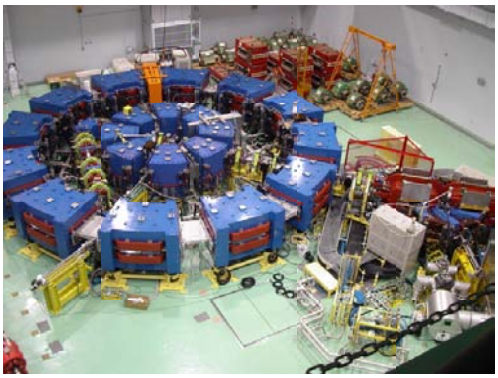


Figure 1: FFAG complex at KURRI.

INTRODUCTION

Kumatori Accelerator driven Reactor Test(KART) project[1, 2] has been approved by the ministry of education, culture, sports, science and technology and started from the fiscal year of 2002. The main purposes of this project is to study the basic feasibility of ADS system and to develop a practical FFAG accelerator as a proton driver for ADS, based on the developments and successes on PoP FFAG accelerators in KEK[3, 4].

In KART project, an accelerator complex which consists of one FFAG with an induction unit for acceleration as the injector and two FFAG with RF as the booster and main accelerators are constructed(Fig. 1). Basic specifications for this FFAG complex are summarized in Table 1. The layout of these FFAG accelerators in the accelerator room is shown in Fig. 1. In near future, this accelerator complex is expected to be served for multi-purpose usages in various fields, such as physics, chemistry, material science and medical applications.

Since this accelerator complex is the first practical FFAG accelerators, many major and minor modifications in the design and equipments have been made during the construction. The control system for this complex is required to accept such time-to-time changes. Our control system should be sufficiently easy for them to use, or develop. While we have to keep such flexibility and easiness, the combined operation with a nuclear fuel assembly requires high reliability and stability towards the control system from the point of nuclear safety.

Table 1: Specification of the FFAG complex at KUR

Beam Energy	25 - 150 MeV
Maximum Average Beam Current	up to 1 μA
Repetition Rate	up to 120 Hz

In our case like other small institutes, the number of technical staffs who support the construction or the operation of the accelerator are limited and they have only limited skills other than their own specific field. As for computing, our average echnical staffs have very limited skills such as web browsing, using mail clients or Microsoft Office suite on Windows environment. Sometimes they can't even setup a Windows PC for network connection. Our control system should be easy for such technicians to use.

To meet such requirements for the present control system, we decide to develope a control system based on LabView and PLC with network capability.

In this paper, the control system for our FFAG complex and some plans for future upgrades of this system are introduced.

SYSTEM OUTLINE

Framework

The framework of the present control system is shown in Fig. 2. The present system is basically the same control flow in the AVF cyclotron at Tohoku University[5], i.e., PLCs are controlled by remote PCs over the fast network. In this architecture, devices and instruments such as power supplies, motor drivers, inputs and outputs of analog signals and digital logic signals are connected to respective

* tanigaki@rri.kyoto-u.ac.jp

PLC modules. Such PLC modules are on TCP/IP network for the communication with the remote PCs, which are served for the human interface and high level sequences.

A big advantage in using PLC is that a wide variety of modules for various kinds of devices are already available as commercial base. One can easily control almost any devices by PLC without special software drivers, while the development of the drivers for devices itself is a kind of important work in other control framework like EPICS.

One of the important feature in this control system is that we positively use the memory of PLC as the database of this control system. PLC accepts various types of commands for memory operation over network, thus we have already possessed a kind of database server in operation on the network without any programming or preparing special database servers. This feature greatly improves the reliability and simplicity of our control system.

Hardware

FA-M3R series by Yokogawa Electric Corporation is used as the PLC in the present control system. This series is well adapted to the network. For example, FA-M3 is the only controller that allows all maintenance over the network except hardware troubles, while other PLCs require to

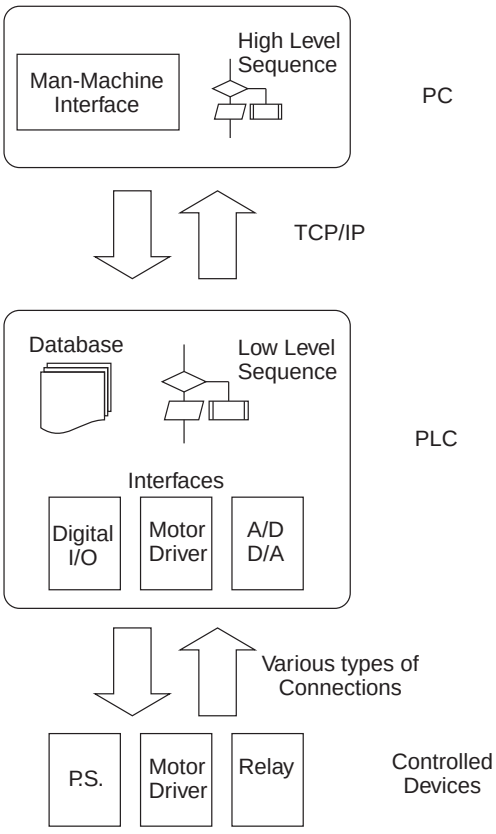


Figure 2: Schematic diagram of the present control system. PLC is responsible for serving the interface between low-level devices and higher PC control software and for maintaining the database of parameters of connected devices.

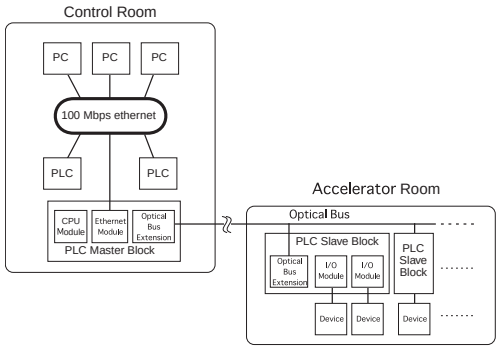


Figure 3: Hardware configuration of the present control system. Each section has one CPU which governs the connected slave modules. All CPUs are placed in the control room and connected over the TCP/IP network.

prepare some direct connections such as USB or RS-232c. FA-M3R also has the capability of bus extension by optical fibers. All modules connected by optical fibers work as if they are one module block, therefore, the configuration of modules is not affected by the physical limitations, such as the location of devices etc. This feature enables CPU to be kept away from the high noise or radiation environment, which always become problems in the control system of accelerators. Another advantage in FA-M3R series is that the backup system for the code. All programs prepared for this PLC is stored on RAM, and this PLC makes a complete copy of these code and perform the check-sum for both codes whenever it is powered on. If either of them has check-sum error, then the error block are automatically restored from the other one. Furthermore, FA-M3R has a built-in backup battery which maintain the data on the memory for ten years without external power supply. With these features, FA-M3R obtains the durability for the memory lost caused by radiation or other reasons without losing the flexibility to the modification of codes. In the usual PLCs, the programming codes are to be stored in ROM for the protection from such memory lost, then one has to prepare ROM every time any modifications are required in the program.

The hardware configuration of the current control system is shown in Fig. 3. All the devices are grouped into several groups based on the hardware configuration such as “ion source” or “booster”, and one CPU is assigned for each group. All the CPUs are placed in the control room to prevent from the electrical noise and radiation damages. Only the slave module blocks, which consists of several interface modules, are implemented close or inside the devices, then connected to the respective CPUs with the optical fiber. A typical implementation of the slave module block is shown in Fig. 4. Each CPU module has its own program to maintain the database of parameters from/to the connected devices. Lower level sequences, such as the hardware protections, are also implemented in PLC. PLC modules are connected to the 100 Mbps ethernet network for the communication with remote PCs over TCP/IP protocol.

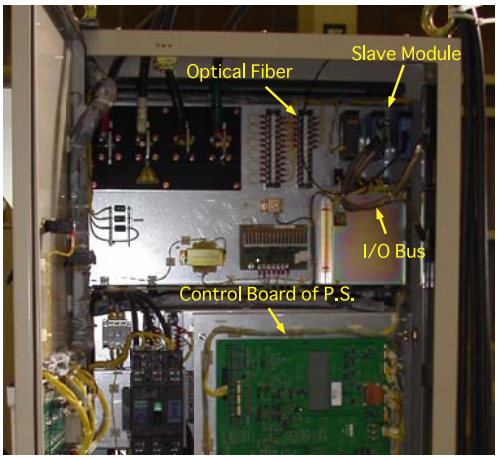


Figure 4: A typical implementation of a slave module in the device. In this case, a slave module block is implemented in the power supply and serves the data communication with the control board of power supply over I/O bus line.

Software

All MMIand higher control sequence are developed in LabView. LabView is known as its easy programming process and support of various operating systems. Developed VIs are basically independent from operating systems and keep good upward compatibility for a long period.

The conceptual diagram of the softwares is shown in Fig. 5.

The remote PC has one or more MMIs and communication VIs which communicate with remote PLCs over TCP/IP. All the communications are initiated by the communication VI, usually every 100 ms. In each communication, all the data on the PLC memory allocated for the parameters from devices is transferred to the PC, then the communication VI translate and store them as global variables in LabView. Any manipulations made by the operator are written into the global variables from MMI VIs, then these values are translated into a set of parameters and transmitted to PLC by the communication VI in each communication cycle. These translations are made by allocation tables described later.

MMIs on PCs can be easily prepared without the special

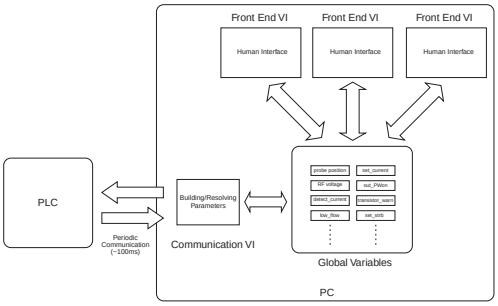


Figure 5: A conceptual diagram of the software on PC.

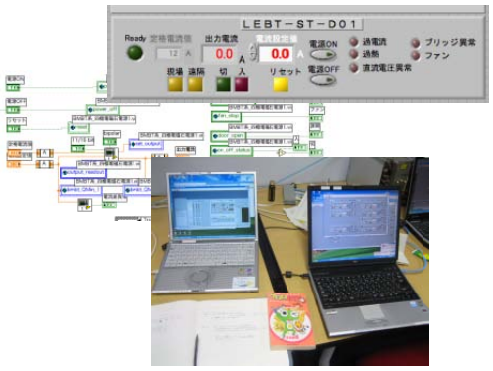


Figure 6: MMI developed on LabView environment. MMI VIs are usually used on conventional Windows laptop PCs.

knowledge on the programming or on the operating system since communication VIs have already expand the parameters as global variables. For conventional users, placing items on the window and wiring them are sufficient to develop MMI and control a device. One can also develop more complicated control sequence in the same way as developing a conventional VI in LabView.

PLC as database server

The parameter set received by the Ethernet module of PLC is expanded to the memory and referred by CPU in a very high speed (less than 0.1 μs). Then the signals comes out from the respective modules. The status of controlled devices are transferred on the memory by the CPU, then the Ethernet module sends the data on the memory upon requests made by remote PCs.

Allocation tables of the parameters on PLC memory are given in text file and referred by communication VIs. This file contains not only necessary information in the translation of data on PLC and VI, such as the name of global variables the assigned address on PLC memory, but also other information such as the physical pin assignment of PLC modules, the conversion parameters and units of data from equipments connected to PLCs or the IP address of the PLC. The allocation tables are originally prepared and maintained on conventional Excel files for the hardware connection between devices and PLCs by the field technician. These files are converted to simple text files then served to communication VIs. An interpretation function of this table format are implemented in communication VIs.

The memory protection features in FA-M3R are very strong, e.g., a built-in backup battery which lasts ten years. This because FA-M3R are supposed to be used under a condition that some memory troubles such as power failure, high level noise may be expected while in the operation. This feature makes our database safe without special protection system.

CURRENT STATUS AND FUTURE PROSPECTS

The development of the control system is basically completed and devices are added as the accelerator is constructed. Conventional laptop PCs are mainly used for serving man-machine interfaces(MMI) and higher control sequence since a lot of field works like test operations of equipments are expected during the construction and test operation. In such cases, these PCs are brought into fields and controls are made over the network. The conventional wi-fi access points are prepared in most of the accelerator building for such laptop PCs. There is no special requirements for PCs other than the network capability and sufficient hardware configuration for LabView. Currently, windows laptop PCs, which are familiar to ordinary technicians, and some MacOS X laptops are mainly used.

As we proceed the test operation of our FFAG complex, a logging system for parameters of accelerators becomes very important. As a candidate for our logging system, MyDAQ[6] developed in spring-8 is recently implemented to our ion source part for evaluation.

Small handy devices are sometimes very convenient in the test operation, so MMI and communication VIs are ported to PDA with wi-fi capability. Thanks to the multiplatform support of LabView, minor modifications such as the rearrangement of user interfaces and performance optimization. An example of such ported VI running on Windows mobile PDA is shown in Fig. 7.

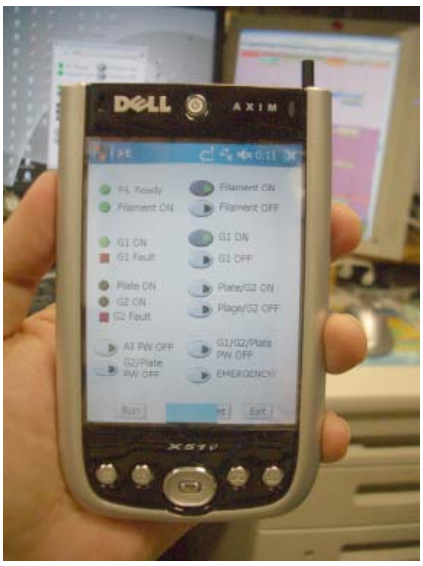


Figure 7: A prototype VI ported to PDA. This is the control panel for an RF power amplifier.

REFERENCES

[1] S. Shiroya, H. Unesaki et al., "Neutronics of Future Neutron Source Based on Accelerator Driven Subcritical Reactor Concept in Kyoto University Research Reactor Institute (KURRI) ", Int. Seminar on Advanced Nucl. Energy Systems toward Zero Release of Radioactive Wastes, 2nd Fujiwara Int. Seminar, Nov. 6-9, 2000, Shizuoka, Japan, *Abstracts* p. 58.

[2] S. Shiroya, H. Unesaki et al., Trans. Am. Nucl. Soc., 2001 Annu. Mtg., June 17-21, 2001, Milwaukee, Wisconsin, p. 78.

[3] M. Aiba et al., "DEVELOPMENT OF A FFAG PROTON SYNCHROTRON", Proceedings of EPAC 2000, Vienna, Austria, p. 581

[4] T. Adachi et al., "A 150MeV FFAG SYNCHROTRON WITH "RETURN-YOKE FREE "MAGNET", PAC 2001, Chicago, the United States, p. 3254

[5] M. Fujita et al., "A CONTROL SYSTEM FOR THE NEW AVF CYCLOTRON AT CYRIC", The 13th Symp. on Acc. Sci. and Tech, (2001) p.106.

[6] Akihoro Yamashita, Toru Ohata, "MyDAQ, A SIMPLE DATA LOGGING AND DISPLAY SERVER", Proceedings of PCaPAC2005, Hayama, Japan

OPERATIONAL EXPERIENCE WITH SYNCHROTRON LIGHT INTERFEROMETERS FOR CEBAF EXPERIMENTAL BEAM LINES

P. Chevtsov, Jefferson Lab, Newport News, VA 23606, USA

Abstract

Beam size and energy spread monitoring systems based on Synchrotron Light Interferometers (SLI) have been in operations at Jefferson Lab for several years. A non-invasive nature and a very high (a few μm) resolution of SLI make these instruments valuable beam diagnostic tools for the CEBAF accelerator. This presentation describes the evolution of the Synchrotron Light Interferometer at Jefferson Lab and highlights our extensive experience in the installation and operation of the SLI for CEBAF experimental beam lines.

INTRODUCTION

Synchrotron radiation (SR) is emitted from relativistic charged particles when they are traveling on curved paths. Because of strong relativistic effects (factor $\beta \approx 1$), the synchrotron radiation is emitted in a very narrow cone in the forward direction tangent to the particle orbit. In other words, each relativistic electron traveling in a magnetic field looks like a moving flashlight giving off synchrotron light in front of itself. The SR has a wide energy spectrum, from infrared to γ rays. Beam size monitors based on the use of synchrotron radiation are not invasive because they do not need to intercept the beam to perform beam size measurements.

A prominent feature of the CEBAF accelerator at Jefferson Lab is a very small ($\sim 2 \cdot 10^{-5}$) relative energy spread of the electron beams provided for the nuclear physics research program. The smaller the beam energy spread, the higher the resolution of the experiment using this beam, and physicists can see more details inside nuclei. Two synchrotron light interferometers have been installed at high dispersion locations 1C12 (Fig. 1) and 3C12 of experimental beam lines to continuously monitor the transverse size and energy spread of the beams for Hall A and Hall C experimental end stations. These high dispersion locations are also equipped with optical transition radiation monitors (OTR), which are invasive and mostly used for the SLI calibration purposes.

Each SLI at Jefferson Lab is a classic wave front division interferometer using polarized quasi-monochromatic synchrotron light.

Synchrotron light generated by the electron beam in a dipole magnet (we call this magnet the principal SLI dipole) is extracted from the beam pipe through a circular viewing port window by means of a rectangular mirror installed in a vacuum chamber ("in-vacuum" mirror). Two additional mirrors guide the synchrotron light through the SLI's optical system. One of them is remotely controlled. We call it the active SLI mirror. The main task of this mirror is to send light on to a CCD camera head through a long (~5 m) plastic pipe, diffraction slits, a narrow band pass filter, a polarization filter, and a CCD camera objective lens, in the direction opposite to the direction of the electron beam. The CCD and optical components are placed in an optical box. The CCD camera is connected to an image processor. A double slit assembly with small slit openings and a predefined set of distances between slits is located right in front of the video camera objective. The assembly is moved by remotely controlled stepper-motors.

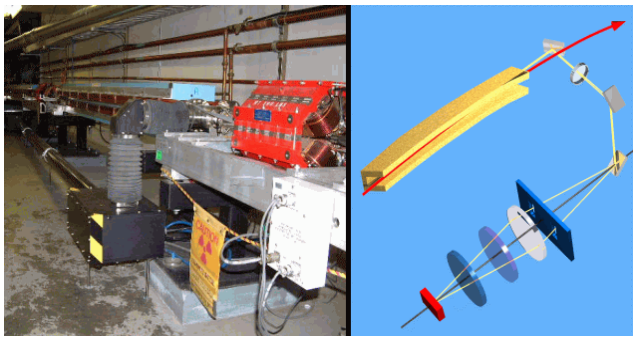


Figure 1. SLI installed at high dispersion location 1C12 at Jefferson Lab and its main optical components.

A limited space and relatively high radiation level in the accelerator tunnel strongly influenced the 3-D SLI design, with main elements placed on two horizontal levels parallel to the ground plane. We note that the upper horizontal plane is determined by the design beam trajectory in the SLI principal dipole and following straight section of the beam line. Consider now a straight line that lies in this plane and intersects the design beam trajectory at a right angle, ~20 cm downstream from the SLI principal dipole. We call this line the SLI reference line. The SLI reference line is very important for the SLI project. In particular, the alignment of all SLI optical components is based on this line, and the viewing port window is installed so that this line goes through its center and is perpendicular to it.

EVOLUTION OF SLI BEAM DIAGNOSTIC SYSTEMS AT JEFFERSON LAB

All basic SLI functions are automated using control and data processing software. This software and SLI components together build up the SLI systems. A very

high resolution (a few μm for beam size measurements) and ability to monitor the beam size and energy spread non-invasively make these systems valuable beam diagnostics tools for nuclear physics experiments. The SLI systems require a lot of work during their installations and support during operations. In this paper, we describe the most critical elements of the SLI systems and their evolution.

"In-vacuum" mirror

We begin with the "in-vacuum" mirror. Its installation must be performed extremely carefully. Based on technical drawings of accelerator components at a high dispersion location, the mirror is positioned in a vacuum chamber so that its center lies on the SLI reference line and redirects the synchrotron light ray originating 0.5 m downstream from the center of the magnet along this line.

Figure 2. Adjustable mount of the SLI "in-vacuum" mirror.

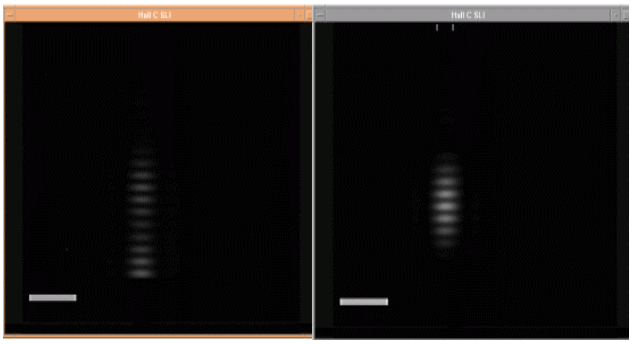


Figure 3. 3C12 SLI interference patterns for incorrect (left) and correct (right) positions of the "in-vacuum" mirror.

In the first (1C12) SLI installed at Jefferson Lab, the "in-vacuum" mirror was fixed. It appeared that we were very lucky in positioning this mirror in the vacuum chamber so accurately that this SLI has been operating without any problem since its installation about five years ago. If we happen to position the "in-vacuum" mirror at a slightly wrong (~0.3 arc degree) angle to the design beam trajectory, a detailed analysis of the geometry of our high dispersion locations shows [1] that the SLI optics can capture two synchrotron light beams, which are almost parallel to each other and are separated by only ~1 cm. The first light beam is emitted from the accelerated electrons in the principal SLI dipole. It produces a primary SLI interference pattern. The second light beam is generated in the upstream dipole magnet, travels in the vacuum chamber in the area of the principal SLI dipole, and reflects from the metallic walls of the chamber. It is this light that we don't want to observe because it produces a secondary SLI interference pattern overlapping with the primary one. The resultant combinations of interferograms are so complicated that it is much easier to modify the SLI design to be able to get rid of all possible secondary synchrotron light sources than to analyze such combinations. All that is needed is to have a possibility to correct small errors in positioning the "in-vacuum" mirror after its installation. That is the reason for making the mirror in the second (3C12) SLI adjustable. The mirror is attached to a remotely controlled mount (Fig. 2) that can be rotated by a specified angle in a horizontal or a vertical plane with a high accuracy (better than 0.1 arc degree). Figure 3 illustrates the working of this mount. It shows the SLI interference patterns for an incorrect and correct position of the "in-vacuum" mirror.

Active SLI mirror

If the "in-vacuum" mirror is positioned properly then the quality of the SLI interference pattern depends on the position of the active SLI mirror, which is adjustable and remotely controlled. To calculate the beam size, the interference pattern is analyzed on the basis of the SLI multi-parameter non-linear data model [2]. One of the necessary conditions for the validity of this model is the requirement of equal light intensities on the diffraction slits. With small fluctuations of the beam trajectory in the CEBAF accelerator around its design orbit, this requirement can always be fulfilled by the adjustment of the position of the active SLI mirror. In the initial SLI design, this mirror was not computer controlled. Its adjustment could only be done manually, by pushing a set of buttons on a keypad. Soon after the first SLI tests, we switched to computer controlled active SLI mirror.

SLI video camera and image processor

SLI systems at Jefferson Lab use an STV CCD camera from Santa Barbara Instrument Group [3]. The integration (exposure) time of this camera can gradually be changed from 0.001 seconds to 10 minutes. The quantum efficiency of the camera is very high. For instance, it is more than 70% for λ_0=630 nm. An electronic cooling system keeps CCD noise extremely low. The camera has its own control box with an RS-232 interface to an external computer.

A choice of a CCD video camera is very important for the SLI project at Jefferson Lab. With relatively low beam currents (~10-100 μA compared to a 10-100 mA range typical for synchrotron light sources) available at high dispersion locations of the CEBAF accelerator the images of synchrotron light interference patterns are dim. In order to make them suitable for data analysis, the camera has to integrate these images over a period of time (~1-10

seconds). Under these conditions, cooling the CCD significantly improves its signal-to-noise ratio.

We note that another practical solution of acquiring dim synchrotron light interference patterns would be to use an intensified camera. Intensifiers work similar to photo multiplier tubes (PMT). They amplify image signals by many (hundreds and even thousands) times, increasing the sensitivity of the camera to almost a single photon level. Unfortunately, at the moment the intensified cameras are 10-20 times more expensive than STV cameras.

The video signal from the SLI camera is fed into Maxvideo 200 [4], which is a pipelined high performance image processing system. The main advantage of the pipeline technology is that the pixel manipulation can be done while the image is being digitized and directed to the image memory. As a result, basic image processing operations can be implemented at the full 30 Hz frame rate of the standard NTSC video signal.

The optimization of the signal-to-noise ratio for the Maxvideo input video signal can easily be done on the basis of a gray scale pattern generator integrated into the STV camera system. The gain and offset parameters of the image processor are adjusted to get in its memory all 256 clean shades of gray provided by the generator. This adjustment is performed regularly (at least once a week) to make sure that the quality of the video signal is good.

SLI control software

The main elements of the SLI control software are the mirror, video camera, and diffraction slit assembly control modules. All these modules are based on the Common Serial Driver/Device Library created at Jefferson Lab [5]. The control is based on the device configuration files, which completely define the communication channels and protocols. The software is easily configurable for any hardware architecture and extremely reliable.

SLI data processing software

Preliminary image processing is done by Maxvideo. The multiplexed software created at Jefferson Lab [6] makes it possible for Maxvideo not only to routinely perform such important operations as masking the pixels outside the region of interest and subtraction of a background but also estimate the beam sizes from two OTR beam images and two SLI interference patterns simultaneously at a up to 10 Hz rate. In the case of the OTR, the beam size is estimated directly from the beam image. For the SLI data, the beam size estimate is based on the visibility (contrast) V of the interference pattern [2].

The final image processing and data analysis are done by an SLI server. It is a typical configuration for high level applications at Jefferson Lab with very powerful calculation and modeling engines. The server runs on a workstation connected to the control system computer network. It takes the information about the beam parameters, SLI components, and the Maxvideo beam size estimates from the accelerator control system and fits the interference pattern with the use of the SLI data model. One of the results of this fit is the visibility V that is used to calculate the beam size and energy spread. The quality of the data fit is controlled by the SLI data model reliability parameter. This parameter is the central part of the SLI data model reliability concept that has been developed at Jefferson Lab [2].

In terms of classical problems of setting up and testing hypotheses, the data model reliability can be defined as the probability of wrongly rejecting the data model on the basis of measured results. Higher reliability of the model means its better consistency with experimental data.

The SLI data model reliability concept allows us to effectively control all basic SLI operations including the calibration of the SLI on the basis of the OTR data and corrections of the active SLI mirror positions when the beam trajectory changes. We note that one of the main results of the SLI calibration is the definition of the optimal (from the data analysis point of view) CCD camera exposure time for each value of the beam current.

CONCLUSION

The implementation of the SLI systems for the experimental beam lines is one of the most successful beam diagnostics projects at Jefferson Lab. We have gained a very valuable experience in the SLI installation and support of all its components in operational conditions. Based on our Common Serial Driver/Device Library, multiplexed Maxvideo software, and data model reliability concept, the SLI systems form powerful beam diagnostics tools at Jefferson Lab. The systems not only routinely monitor the transverse sizes and energy spread of electron beams in a wide range of beam intensities but also help identify beam trajectory (energy) problems in the accelerator.

ACKNOWLEDGMENTS

The author is very thankful to H. Areti for his interest in the SLI project and support.

REFERENCES

[1] P. Chevtsov,"Synchrotron Light and its Interferometry at CEBAF Beam Lines", Jefferson Lab Tech. Note JLAB TN-06-033, 2006.

[2] P. Chevtsov, "Automated Image Quality Optimization for Synchrotron Light Interferometers", ICALEPCS 2005, Geneva, Switzerland, 2005.

[3] www.sbig.com

[4] www.datacube.com

[5] P. Chevtsov, S. Schaffner, "Information-Control Software for Handling Serial Devices in an EPICS Environment", ICALEPCS-2001, San Jose, CA, USA, 2001.

[6] P. Chevtsov, et al., "Multivideo Source System for Beam Diagnostic Applications", PCaPAC 2000, DESY, 2000.

BEYOND PCS: ACCELERATOR CONTROLS ON PROGRAMMABLE LOGIC

M. Pleško, K. Žagar, A. Hasanović
Cosylab, Ljubljana, Slovenia

Abstract

The large number of gates in modern FPGAs including processor cores allows implementation of complex designs, including a core implementing Java byte-code as the instruction set. Instruments based on FPGA technology are composed only of digital parts and are totally configurable.

Based on experience gained on our products (a delay generators producing sub-nanosecond signals and function generators producing arbitrary functions of length in the order of minutes) and on our research projects (a prototype hardware platform for realtime Java, where Java runtime is the operating system and there is no need for Linux), I will speculate about possible future scenarios: A combination of an FPGA processor core and custom logic will provide all control tasks, slow and hard real-time, while keeping our convenient development environment for software such as Eclipse. I will illustrate my claims with designs for tasks such as low-latency PID controllers running at several dozen MHz, sub-nanosecond resolution timing, motion control and a versatile I/O controller - all implemented in real-time Java and on exactly the same hardware - just with different connectors.

INTRODUCTION

When performance is paramount, one of the techniques a skilled electronics engineer would use is *reconfigurable computing* [1]. In terms of performance, this approach is the second best to what is achievable with commercially available electronics, only surpassed by *application specific integrated circuits* (ASIC).

Reconfigurable hardware has several advantages over the fixed wiring in an ASIC. Firstly, it allows for quick and inexpensive prototyping cycles. If a defect in design is uncovered, it can be quickly fixed, and the hardware can be reprogrammed in a matter of minutes.

Application-specific reconfigurable hardware can also be much faster than general-purpose processors, even though processors operate at higher clock rates. Thus, performance improvements are significant (order of magnitude improvement is not uncommon).

Most commonly used reconfigurable devices today are field-programmable gate arrays (FPGA) and complex programmable logic devices (CPLD). CPLDs are less expensive, but do not offer as much performance and programming capacity as FPGAs. Prices of these devices range from several Euros for CPLDs, to several hundred Euros for high-end FPGAs. Market leaders are Xilinx and Altera.

Pure programmable hardware implementations might be costly in terms of development effort, however. Some problems are inherently difficult to solve in hardware using programming languages such as VHDL and Verilog. To this end, FPGA vendors offer generic processor cores. In some cases, the cores are available in form of VHDL code (e.g., Altera NIOS or Xilinx PicoBlaze), whereas in higher-end FPGAs multiple powerful cores are fixed on the chip (e.g., Xilinx Virtex II includes PowerPC cores).

With generic processor cores, the development can be split in two parts: performance intensive part is implemented in hardware, and the complex part in assembler or C.

However, development in a C-like language is still cumbersome. Illegal use of memory, buffer overruns, memory leaks, relatively long compilation times, portability issues and structured programming approach contribute to decreased efficiency, which is in some cases even 2 to 10 times smaller than one achievable with higher-level programming languages, such as Java.

REQUIREMENTS

This section lists some of the requirements that a hardware Java platform would have to meet in order to retain high-level of development efficiency.

Standard Java constructs should be retained. I.e., no new keywords should be introduced into the language. This way, existing tools for Java development could be leveraged, such as high-productivity integrated development environments (IDEs, e.g., Eclipse or NetBeans), compilers, byte-code manipulation tools and code verifiers.

1. **Hardware** should be composed of **modules**. Each module would consist of the hardware part (templated VHDL files) and software part (configurable drivers). When a module would be instantiated, the software drivers would be automatically configured for the instantiated hardware (e.g., matching bus addresses, IRQ numbers, etc.).
2. **Static (compile-time) checking** should be possible. For example, it should be impossible to overlap register addresses of modules on a bus. Ideally, the register addresses would be assigned automatically.
3. **Support for debugging**. A debug console should be available through a serial port. In addition, Java virtual machine should support remote debugging using existing tools. JTAG diagnostics of hardware should also be possible.

4. **Field upgrades** of hardware and ROM software should be possible.

One-size-fits-all board: ideally, a general-purpose board design would exist, so that boards would not have to be developed for each application specifically. A modular board composition (IndustryPack, PC/104, VME) is a good approach to achieve this.

EXAMPLE APPLICATIONS

Nanosecond Resolution Timing

In particle accelerator controls, sub-nanosecond resolution timing is sometimes required due to high speed of particles whose orbit needs to be controlled. A particular application called for a controllable delay generator, capable of producing output signals that are delayed relative to a trigger signal for amount of time in the order of a nanosecond. Figure 1 shows an example of a trigger signal and the resulting output signals, which are delayed by t_A and t_B, respectively.

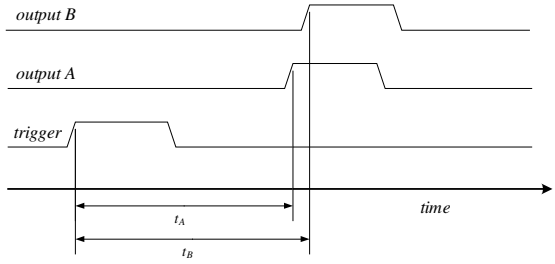


Figure 1: signals of a delay generator.

To this end, FPGA with a phase locked loop (PLL) can be employed. The PLL is capable of multiplying the clock frequency by a given factor. Thus, if a 500MHz external clock is used, the PLL can multiply it by 4, achieving a 2GHz clock (0.5 ns temporal resolution). The jitter of this clock is also very small (in the order of 50 ps), which makes FPGA technology a good candidate for this application.

Since the delays t_A and t_B induced by the FPGA-based delay generator would have to be externally controlled (ideally through a SCADA-like system using a computer network), a pure VHDL solution is no longer a feasible option. Therefore, a co-design approach depicted in Figure 2 is a reasonable alternative. Here, a processor is monitoring communication over Ethernet, implementing TCP/IP or other network stacks required by the SCADA system, implementing their respective protocols. The processor then converts the requests from Ethernet to configuration for the delay generator through a bus internal to the FPGA. The processor, Ethernet MAC layer and delay generator are thus all contained in a single FPGA chip, whose input pin is a trigger, and whose output pins are correspondingly delayed. (Apart from these pins, also pins for reset, clock, ground, power supply, etc., are required).

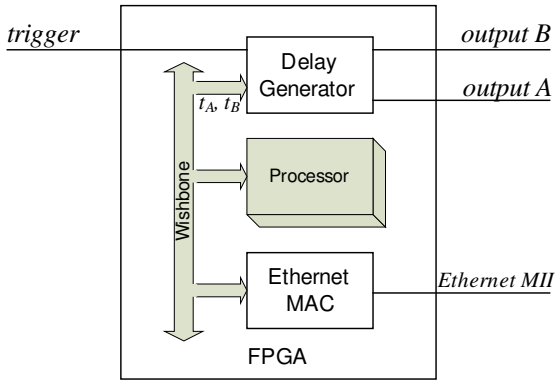


Figure 2: block diagram of a delay generator.

Versatile I/O Controller

In automation applications, integration with many kinds of devices is required. The devices are equipped with various control interfaces, ranging from analog current, via serial interfaces (e.g., RS-232, RS-485, etc) to more sophisticated busses, such as *General Purpose Input/Output Bus* (GPIB).

In some cases, for example control of particle accelerators, the number of devices under control is large, and there are tens of thousands of process variables that need to be controlled or monitored. Consequentially, the density of I/O channels is high, and entire racks are devoted to front-end control equipment (also called *Input/Output Controller*, IOC). This control equipment is responsible for performing simple tasks, such as communicating with devices with their respective protocol, initializing devices, converting the values returned by devices from raw to engineering units, etc. One of the control equipment's most important responsibilities is to make the connected devices available to a SCADA system via a computer network.

Implementing an IOC in programmable hardware might well be a very economic and efficient approach. The inputs and outputs of these controllers are then bound to pins of the FPGA, and from there to the IOC's board, where transceivers implementing the physical layer of communication are placed.

The pin count of FPGAs is fairly large – several hundred pins are available for application-specific purposes (e.g., Altera offers FPGAs from 484 to 1508 pins). One UART serial line requires 4 signals, which means that physically more than 100 serial connections could be handled by a single FPGA.

Since voltage levels of FPGA's pins are not arbitrary, the board would require transceivers to implement the physical layer of the communication stack. Also, some I/O controllers are not easily available or are difficult to implement. One such example is GPIB – in this particular case, integrated circuits are available, which can be integrated with FPGA through a standard bus (e.g., National Instruments' TNT5002, which uses PCI bus).

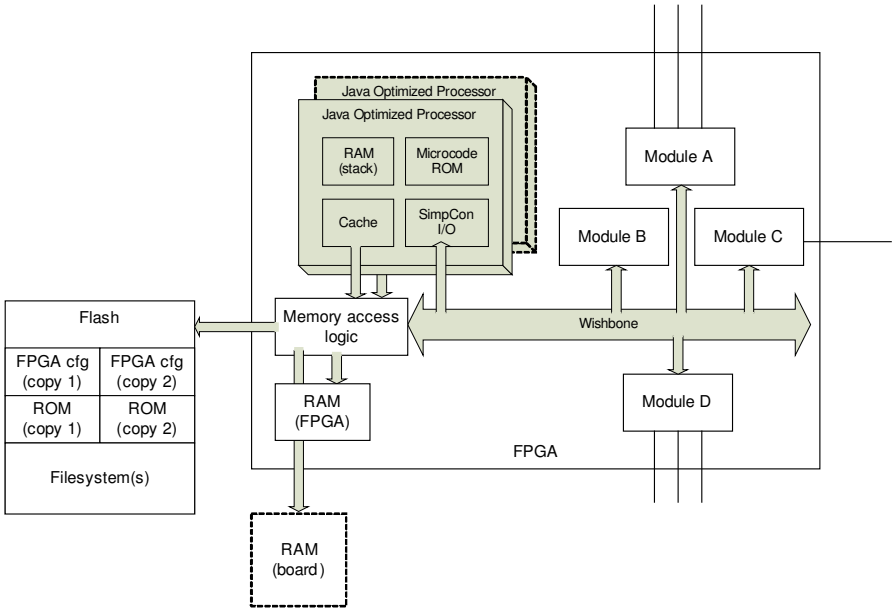


Figure 3: Overview of the hardware architecture..

Well though-out modular design of the board would allow re-use of the same design for various I/O configurations. Such a board would either contain a very large pin bank to which connectors would be attached, or feature an extensible interconnect bus (VME, IndustryPack, etc.).

ACHIEVING HARD REALTIME

In automation**, hard real-time interlocks** are frequently a requirement. When an interlock is triggered, a reaction (e.g., a shutdown or switching-off of an output) must commence immediately. In FPGAs, such interlocks can be implemented directly in hardware. If they are implemented asynchronously, the reaction time is only limited by propagation delays, and doesn't even have to wait till the next period of the system clock. Figure 4 shows a circuit of such an implementation of an interlock: whenever the interlock signal is grounded (becomes 0), the output becomes 0 without waiting for logic or clock. The response time is thus even shorter than 1 nanosecond.

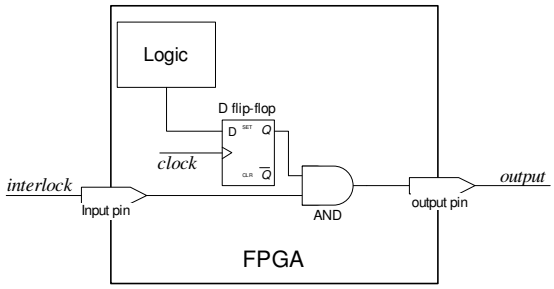


Figure 4: Digital circuit implementing an interlock.

VHDL code for this circuit (Listing 1) shows a VHDL design pattern where an interlock is implemented without affecting the rest of the logic – thus, the safety aspect of the system can be introduced in a design systematically, without affecting the design of the logic.

```
-- synchronous implementation of the logic
process(clock)
begin
      if rising_edge(clock) then
      begin
            logic <= …;
      end if;
end process;

-- asynchronous handling of an interlock
output <= '0' when interlock='0' else logic;
```

Listing 1: VHDL code of an interlock.

HARDWARE ARCHITECTURE

The architecture follows an established pattern: it features a CPU, memory/storage (on-chip and off-chip), various modules, and a bus that interconnects all of the components together (Figure 3).

For the CPU core, we propose using a standard, possibly open, implementation. Cores implementing the Java virtual machine specification in hardware already exist, for example *Java Optimized Processor* (JOP, [2]).

A good candidate for the bus is Wishbone [3]. Wishbone is a flexible, yet simple bus for interconnection of cores within a programmable chip. Since many hardware components (such as Ethernet MAC implementations) exist that offer a wishbone interface, supporting wishbone would allow leveraging these implementations.

CONCLUSION

In this paper, we have tried to illustrate the advantages and application potential of programmable logic. Since this approach offers a lot of freedom, a well thought-out architecture should be agreed upon to prevent unnecessary divergence of efforts and allow re-use of components and methodologies. Ideally, the approach would also leverage the standard integrated development

environments, reducing the learning curve of engineers making use of the technology.

REFERENCES

[1] K. Compton, S. Hauck. “Reconfigurable Computing: A Survey of Systems and Software”, ACM Computing Surveys, Vol. 34, Np. 2, June 2002, pp. 171-210.

[2] Martin Schöberl. “JOP. A Java Optimized Processor for Embedded Real-Time Systems”, PhD thesis, Vienna University of Technology, January 2005, http://jopdesign.com.

[3] opencores.org. “WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores”, revision B.3, September 2002.

MONO FOR CROSS-PLATFORM CONTROL SYSTEM ENVIRONMENT*

H. Nishimura[1] and C. Timossi[2], LBNL, Berkeley, CA 94720, U.S.A.

Abstract

Mono is an independent implementation of the .NET Framework by Novell that runs on multiple operating systems (including Windows, Linux and Macintosh) and allows any .NET compatible application to run unmodified. For instance Mono can run programs with graphical user interfaces (GUI) developed with the C# language on Windows with Visual Studio (a full port of WinForm for Mono is in progress). We present the results of tests we performed to evaluate the portability of our controls system .NET applications from MS Windows to Linux.

.NET PLATFORM AND CONTROLS

Windows in an Accelerator Environment

Microsoft Windows is widely used as a platform for GUI based applications used for machine operations and physics studies. Windows is also a useful platform for instrumentation controls because of the wide industry support of specialized drivers for it. For these reasons Windows has become an essential platform for accelerator and beamline control systems.

The fundamental programming API for developing Windows software is changing from Win32 to .NET. The .NET framework has been evolving since .NET 1.0 in 2002, and reasonable backward compatibility with existing Windows software APIs such as Win32 and Active/X (COM) has been maintained. During this time we have been adapting our applications for the .NET framework on Windows.

EPICS on Windows at ALS

Our accelerator control applications use the Channel Access (CA) layer of EPICS [1] to access accelerator controls data. To support the widest variety of development tools on Windows (e.g. Delphi, C++ Builder, Visual Basic and LabView), we package CA as an ActiveX Control we call SCACOM [2]. This control is a thin wrapper around another library, Simple Channel Access (SCA)[3], that was developed to ease CA client development at the ALS [4].

Although .NET programs can use ActiveX controls directly, there is an advantage to repackaging the control as a .NET assembly. ActiveX is only available on Windows platforms whereas .NET was designed to be portable to other platforms. So, a .NET application - even a GUI application using WinForm - at least has the potential to run unmodified on a non-Windows OS. We've named this new assembly: SCA.NET [5]. In fact, we did much more than re-package SCACOM. We decided to spend some effort recoding some of the routines (in C#) to make better use of CA, thus improving data access performance.

*Work supported by the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

[1] H_Nishimura@lbl.gov. [2] CATimossi@lbl.gov

MONO AS .NET ON LINUX

Mono for Cross Platform Support of .NET

Mono [6] is an implementation of the .NET Framework originally developed by Ximian which is now under Novell. It can be run on multiple operating systems (including Linux, Mac OS X, Solaris, BSD, and Windows) on multiple hardware platforms (s390/s390x, SPARC, PowerPC, x86, x86-64, IA64 and ARM). We report on Linux running on x86 based PCs in this paper.

Compatibility

The Microsoft .NET Framework for Windows has evolved from version 1.0 in 2002, to 1.1 in 2003, to 2.0 in 2005 (version 3.0 will be delivered on Vista). Not surprisingly, Mono has lagged behind Microsoft and is currently delivering version 1.1. In addition to the basic framework, Mono also includes support for ADO.NET for database access and WinForm for GUI development.

In our experience, non-visual classes, such as ADO.NET, have been well supported on Mono.

On the other hand, GUI programming using WinForm is behind that in .NET 1.1 on Windows. When we use graphical libraries from third parties, we need to take extra steps to assure their availability/compatibility on Mono. For example, we use a popular open-source library ZedGraph[7] for plotting and charting. Its newest version 5.0 is for .NET 2.0. However, the previous version for .NET 1.1 required only minor modifications to run on Mono.

Third party library support can also be an issue. Generally speaking, these .NET libraries, which applications need to access with the Platform Invoke interface, are now moving to managed code rather than unmanaged DLLs. However, this move to managed code often occurs together with the migration to .NET 2.0, which is only partially supported on Mono.

At the time of this conference in October 2006, the version of Mono is at 1.1.17. This version basically covers .NET 1.1 and some of the new .NET 2.0 features. Better compatibility is expected with the release of Mono 1.2 and 2.0 in the near future.

Mono as Runtime Environment

Mono becomes a runtime environment for the .NET 1.1 programs developed on Windows, including those made using WinForm. In principle, they should run on Linux with Mono without rebuilding as long as run-time libraries are available. However, it is not unusual to

modify and rebuild such programs to alleviate minor incompatibilities.

To start a .NET application with Mono:

```
$ mono WinApp.exe
```

where WinApp.exe is a .NET application developed either on Windows or with the Mono development tools.

Mono as Development Environment

Mono includes a C# compiler (MCS) that enables .NET development on non-Windows platforms. Although there are more than 10 programming languages already available on Mono [8], including Java, Visual Basic.NET and Python, we currently focus on C# on Mono.

MonoDevelop [9] is the integrated development environment (IDE) for Mono, running primarily on Linux. It can import a Visual Studio 2003 solution containing .NET 1.1 projects in C# developed on Windows. It supports GUI development with several GUI tool kits including GTK# with visual designer. However, WinForm is not currently supported in this manner.

MONO FOR EPICS CLIENTS ON LINUX

SCA.Net for Mono

SCA.NET wraps the Channel Access shared libraries as a .NET assembly that calls into CA using the Platform Invoke API (also known as P/Invoke). For example, on Windows:

```
public unsafe class Ca
{
   . . .
   [DllImport("ca.dll")]
   public static extern
         short ca_field_type (IntPtr ChanID);
   . . .
}
```

On Linux, we would expect to need to replace the reference to “ca.dll” with “ca.so” (the shared library for Linux-x86). However, if this class is built with “Any CPU” option, this is not required. It picks up “ca.so” properly at runtime on Linux. Therefore, there is no need to modify the source code of SCA.NET.

Here is an example of a client program running on Windows XP (Fig.1) and Linux (Fig.2); there is no need for source code changes. It reads the beam current and beam locations (X, Y) at 4 locations through SCA.NET. Its binary is identical on both platforms. Here ca.dll is in the PATH on Windows and ca.so is in the LD_LIBRARY_PATH on Linux.

Thus, by carefully limiting the use of WinForm 1.1 controls for GUI programming, Linux is seamlessly supported at run-time for .NET 1.1 programs with EPICS access.

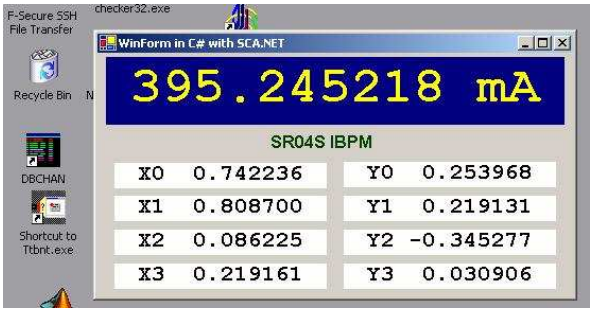


Fig.1. EPICS Client Program on Windows XP

Fig.2 EPICS Client Program on Linux

AKNOWLEDGEMENTS

The authors thank A. Biocca and D. Robin for their support, T. Scarvie for useful advices, and C. Ikami and T. Kellogg for their technical support.

REFERENCES

[1] L. R. Dalesio, et al., ICALEPCS '93, Berlin, Germany, 1993.http://www.aps.anl.gov/epics

[2] C. Timossi and H. Nishimura, IEEE PAC’97, 0-7803-4376-X/98, p805, 1998

http://www-controls.als.lbl.gov/epics_collaboration/sca/win32

[3] http://www-controls.als.lbl.gov/epics_collaboration/sca

[4] LBL PUB-5172 Rev. LBL,1986

A. Jackson, IEEE PAC93, 93CH3279-7(1993)1432

[5] H. Nishimura and C. Timossi, PCaPAC 2005, Hayama, Japan, 2005.

[6] http://www.mono-project.com

[7] http://zedgraph.org

[8] http://www.mono-project.com/Languages

[9] http://www.monodevelop.org

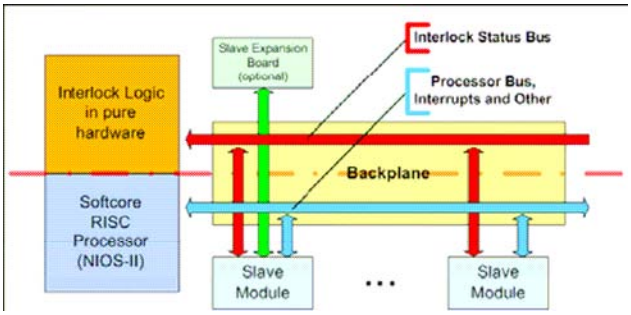
A configurable Interlock System for RF- Stations at XFEL

M.Penno, T. Grevsmühl, H.Leich, A. Kretzschmann, W.Köhler,
B. Petrosyan, G.Trowitzsch, R.Wenndorff, DESY, Hamburg, Germany

INTRODUCTION

The European XFEL-Project[1] requires for its superconducting cavities at the acceleration section up to 40 RF-Stations. Each RF-Station requires an interlock system, which has to prevent any damage from the cost expensive components of the RF-Station.

INTERLOCK CONCEPT

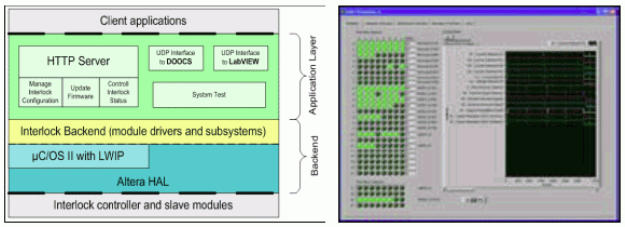


The main advantage of the interlock system is its interlock-logic, which is completely implemented in hardware and doesn't depend on software processes during operation. That is a primary requirement of the concept. Only initialization and self-test at power-up are accomplished by software.

INTERLOCK I/O MODULES

The system holds several I/O modules, which are connected to several input and output signals of different types. The signals are connected via distribution panels to the components of the RF-Station. All modules communicate there signal status periodically to the interlock logic over the status bus to the interlock controller module.

INTERLOCK CONTROLLER



The interlock controller holds a FPGA, which contains the interlock logic and a NIOS2 Processor, where the software runs on. The software performs a self-test on power-up, where all hardware components of the interlock get checked if ready for operation. Also the system provides several interfaces over network to communicate with the control-system DOOCS and diagnostic tools. The tools are used to record detailed information and help to identify noise and error sources.

FUTURE PLANS

Because the NIOS processor doesn't have a MMU[2] (which affects reliability and deterministic behaviour) and existing software is difficult or impossible to port to NIOS2, we are looking forward to switch to a more popular platform like an AMD Geode x86 Processor which also brings along better linux support. That will allow us to run easily TINE and/or DOOCS server on it and other software that belongs to the interlock system.

[1] XFEL – European X-ray free-electron laser project
[2] MMU – Memory Management Unit

MAGNETIC FIELD MAPPING (MFM) SYSTEM FOR SUPER CONDUCTING CYCLOTRON (SCC) IN VECC

Sarbajit Pal, Anindya Roy, Tanushyam Bhattacharjee, N. Chaddha, R. B. Bhole, S. Dasgupta
Variable Energy Cyclotron Centre, DAE, 1/AF Bidhannagar, Kolkata 700 064,
e.mail: sarbajit@veccal.ernet.in

Abstract

The median plane magnetic field of the SCC magnet (Peak field 5.8T) has been measured over its operating range and upto 29 inch radius. The complete map of 360 degree at one degree interval is obtained in less than 100 minutes measuring nearly 100K field values with radial interval of 0.1 inch.

The software of the PC based mapping system works in a Client-Sever environment, maintaining TCP communication between the mapping mechanism controller station as Server and the operator's Console station as Client. The client-server developed using Labview and Windows console program respectively, provides a reliable and easily modifiable GUI and also fast hardware control from the PCs' running Windows-XP. The server program supports remote client control of motors and control and operation of digital integrator & NMR jig along with the collection and logging of measured data. The client program also supports online preliminary display and analysis of field data.

The results indicate the correctness of the magnet assembly and measurement system. After coil-centering and shims placement the maximum first harmonic in the fields obtained are less than 16, 12 and 7 Gauss respectively in central, middle and extraction region at all magnet excitation.

INTRODUCTION

The Magnetic Field Mapping (MFM) system comprises of a hardware system and a control & data acquisition software [1]. A search coil and Digital integrator combination are used to measure the difference in field between the center and any other point of the magnet [2]. The schematic diagram of the system is shown in the Fig 1.

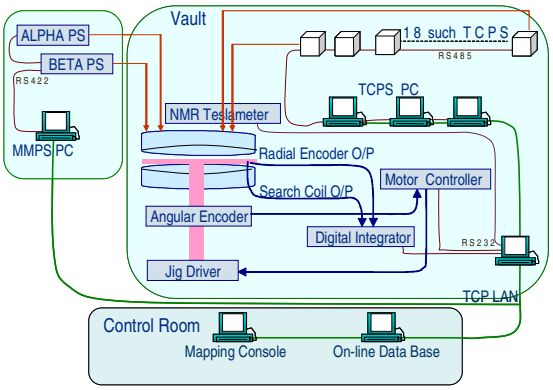


Fig. 1: The schematic diagram of the system

HARDWARE SYSTEM

The hardware system consists of MFM jig, NMR Gauss meter, Digital integrator, Measurement PC and Console PC. The MFM jig comprises of a centrally supported search-coil carrying arm. The radial movement range of the search coil is 29inch. The radial position is sensed by an optical encoder (U. S. Digital) traveling over a linear strip (360 LPI). One out of every thirty-six pulses from the encoder is used to trigger the digital integrator unit to read and integrate the search coil output between the triggers. The angular position of the search coil carrying arm is determined by absolute rotary Inductosyn encoder (256/2 pole, 128/1 speed, 8.15 inch stator O.D.). The standard accuracy of 1.7arc Sec is obtained using two dual channel preamplifier (219200) and AWICS converter board (220500). A microcontroller based interface module, developed in house, reads the angular position from the AWICS board online.

The Measurement PC is placed at vault near the main magnet and the Console PC is placed at the main control room. Both PCs are connected to the dedicated LAN commissioned at Super-conducting cyclotron (SCC) building. The Measurement PC is connected with Digital Integrator, NMR Gauss meter and angular encoder on serial ports. Two Animatics smart motors (SM2337DT) connected in RS232 daisy-chain architecture to the measurement PC, control radial movement and angular position of the search coil.

CONTROL & DATA ACQUISITION SOFTWARE

The control & data acquisition software has two independent modules. The Measurement Controller module, running on Measurement PC, communicates to control and acquire data from digital integrator, NMR gauss meter and search coil positioning system. It also communicates with the Console PC to receive command and transfer all integrated field values of a radial run in bulk mode. The MFM User Interface module running on Console PC, enables users to control the complete process of field mapping comprises of acquiring, storing and on/off-line analysis of the data. This module also communicates with centralized database server to gather related power supplies, cryogenic and other environmental parameters.

Features

The MFM system is designed to be fully automatic to reduce human interference resulting into a minimum measurement time. To meet this requirement, the

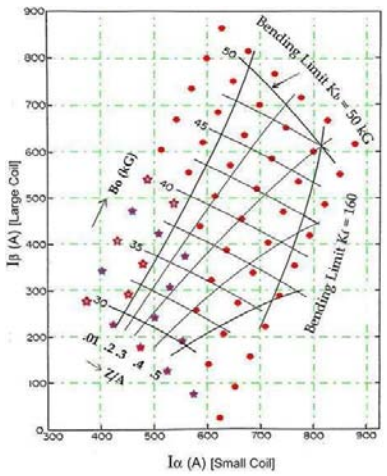


Fig.2. Measurement grid in I_α , I_β plane.

Measurement Controller (MC) and MFM User Interface (MUI) softwares are developed to cater the following salient features.

- Faithful communication, control and acquisition of data from the individual modules, i.e. Digital integrator, NMR Gauss meter, Smart motors and Angular encoder.
- Generation of warning/error messages for unacceptable events during mapping.
- Full remote access and control of the MFM system from the SCC main control room.
- Acquisition, display and storage of other sub-systems data.
- On-line analysis of the acquired data.
- User-friendly operation interface.
- TCP socket communication between MC server s/w and MUI client s/w.

Measurement Controller Software

A multithreaded C module implemented using Windows API performs following three tasks simultaneously.

- Receive command from client program and decode the command to invoke appropriate job.
- Execute the current job.
- Execute a watchdog timer independently to monitor any unacceptable event and notify the client.

The program is divided into following five modules.

Main server thread initializes the search coil movement and positioning system, opens protected TCP socket to communicate with the MFM User Interface. MUI has a predefined IP address to receive command and accepts command from the client to invoke the appropriate task. It also generates error message in case of any failure.

- Digital Integrator thread issues commands to digital integrator module for controlled data acquisition.
- NMR Gauss Meter thread reads online NMR Gauss meter output.
- Angular movement thread positions the search coil carrying arm at desired angle by communicating with angular encoder and smart motor driver.
- Watchdog as a monitoring thread with a preset time out, facilitates to come out of any deadlock situation.

MFM User Interface Software

This software developed in LabVIEW 6.1 incorporating multi-threaded architecture, performs following three tasks simultaneously [3].

- Communication with Measurement Controller s/w for automatic and manual mode of operation, control and monitoring the MFM procedure and field data acquisition and storing.
- Transaction with a centralized Oracle Database server through Microsoft ActiveX Object in SQL to read and display different subsystem (Main magnet power supply System, Trim coil power supplies system, Cryogen delivery system) parameters to ensure the magnetic field during the mapping.
- Online 2D visualization of the acquired field data for comparing the actual and theoretical profile the magnetic field.
- Offline Fourier analysis of MFM data stored in database for the analysis of azimuthal field modulation.

RESULT

Extensive magnetic field mapping has been carried out at different main coil excitations (Fig.2). Three-fold symmetry dominated magnetic field distribution is a characteristic feature of the three-sector geometry of SCC as shown in Fig.3. Deviation from perfectly three-fold symmetry, arising out of manufacturing tolerances and assembly errors, is shown in a contour plot of data at I_α= 300A, I_β =300A (Fig.4). The blue contour is of zero deviation, i.e. perfectly three fold symmetric data points, the red contours are of positive deviation with 10 gauss step and the green contours are of negative deviations with same step. This includes all harmonics deviating from perfect 3-fold symmetry. The radial distribution of B_{av} for different current settings (I_α , I_β) is shown in Fig.5. Iron shims were added to remove unwanted dips in the average iron field distribution at several radii (Fig.6). The distribution of 1st harmonic field at different stage of correction is shown in Fig.7 by map1, map2, and map3. The analysis of the measured magnetic field data and the subsequent correction of the magnet have improved the field quality.

CONCLUSION

The computerized MFM electronic set up is installed inside near Super-conducting magnet. The complete search coil carriage movement and positioning system comprises of the motors, encoders with their drivers and supported electronics, are tested successfully. The starting point of measurement for radial scan is achieved by precise positioning of the linear scale with respect to the encoder head near the central region. The phase and frequency adjustment of the angular encoder electronics is done for extending the cable length running from mechanical encoder head to the control electronics kept at control rack near the measurement PC.

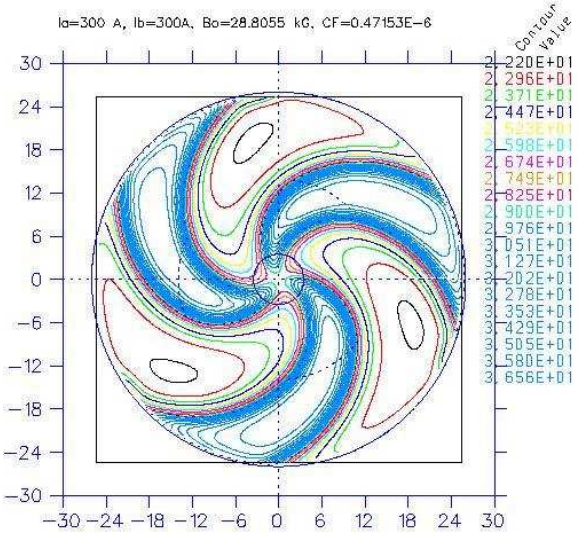


Fig.3. Isogauss contour plot of measured field.

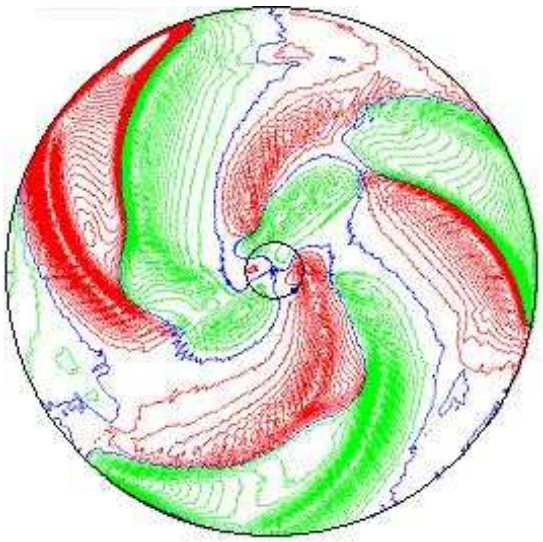


Fig.4. Deviation from 3-sector average

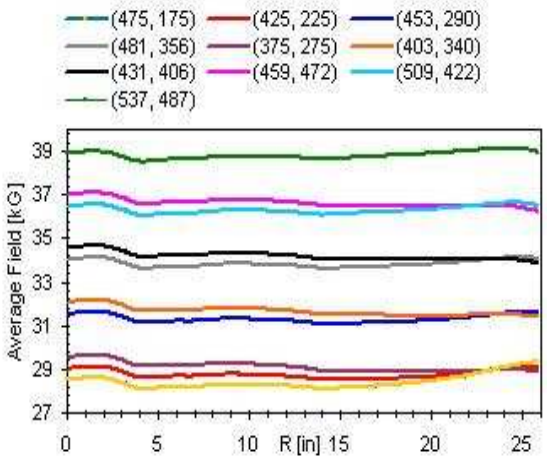


Fig.5.: (a) Average field profile for different (I_{α} , I_{β})

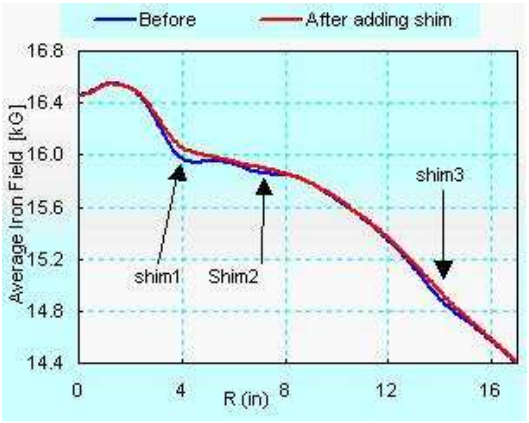


Fig.6. B_{av} correction by adding shim

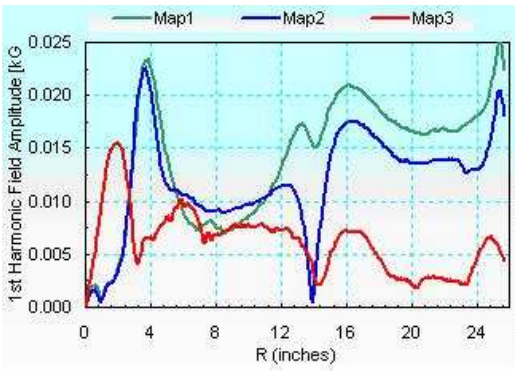


Fig.7. Minimization of 1st harmonic field profile

The mean deviation of the pulse count received from the radial encoder for complete radial movement of the search coil is ±2 count i.e. 20 ppm errors. The data analysis comprises of the following major activities: (i) processing of the measured raw data, (ii) reproducibility check up, (iii) finding the magnetic symmetry axis by minimization of second harmonic component, (iv) correction of average iron field distribution by adding iron shims and (v) minimization of first harmonic component by shimming of iron is successfully completed.

ACKNOWLEDGEMENT

The authors wish to acknowledge Mr. M. K. Dey and his group members for providing useful suggestions, data analysis and plots described in this paper.

REFERENCES

[1] Sarbajit Pal, et al, Development towards magnetic field mapping system, Indian Particle Accelerator Conference-2003, Feb 3-6, 2003, Indore, India.

[2] L.H. Harwood and J. A. Nolen Jr, Plans for Magnetic Mapping of the NSCL K800 Cyclotron Magnet, CHI996-3/84/0000-0101.

[3] F. Marti and P. Jhonson, A LabView based Cyclotron Magnetic Field Mapping System, Proc Int. Conf. On Cyclotron and their Applications, Caen, 1998.

MCS-8 EIGHT AXIS EMBEDDED MOTION CONTROL SYSTEM

G. Jansa, R. Gajsek, M. Kobal, Cosylab, Ljubljana, Slovenia

Abstract

MCS-8 is an 8 axis motion control system for stepper motors, servo drives and a variety of special drives. Central to the controller is the Delta Tau Turbo PMAC 2 programmable multi axis controller. It incorporates a full PLC specifically designed for positioning and control functions.

We have developed a layer on top of the generic and complex Turbo PMAC 2, making the MCS-8 very easy to use as a stand alone box in a control system. Software running on embedded controller can range from EPICS, SPEC to local SCADA system with an Ethernet and RS232 interfaces.

MCS-8 is capable of controlling even the most complicated motion such as Stewart-Gough platform (Hexapod). Hexapod offers six degrees of freedom positioning system with sub-micrometer precision and repeatability. PMAC controller is used to calculate kinematics and EPICS for end user interface. Important features are user selectable point of rotation in space and point to point scanning of all six axes.

INTRODUCTION

In particle accelerators there are many applications that require controlling of motors. These applications range from controlling injection devices in storage ring, such as undulators and wigglers, to components in beamlines which can be as simple as slits or complicated devices such as Double crystal monochromators (DCM) or mirrors. Especially DCMs usually requires controlling of different types of motors such as servo motors with dual feedback for velocity and position, stepper motors, or even nano or pico motors for fine positioning of crystals. MCS-8 is built in a modular way so that it supports any combinations of motors.

Figure 1: MCS-8

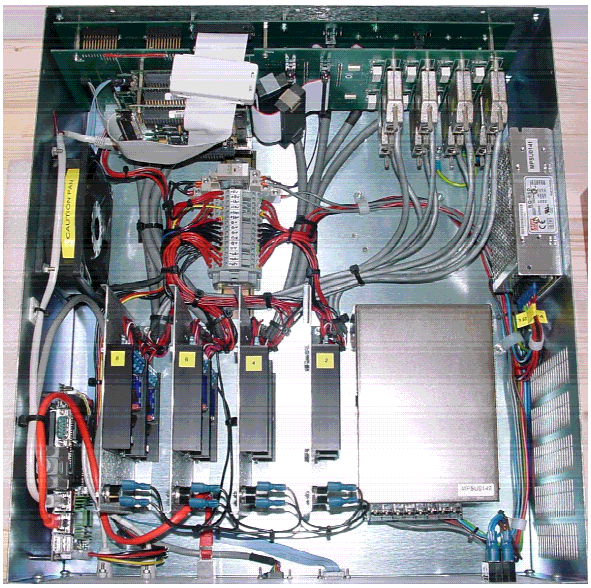


Figure 1: MCS-8 inside

Main hardware components of MCS-8 are Turbo PMAC 2, embedded computer (microIOC) and power drivers. MCS-8 software is divided to low level software residing on Turbo PMAC 2. Basic motion control includes programs for single axis point to point moves, home reference search and various housekeeping tasks, programs synchronize motion of multiple motors can be achieved. Robotic applications control can be performed using kinematics calculations. In recent projects we used EPICS as an interface to PMAC for controlling most devices used in synchrotron light beamlines such as slits, mirrors, DCM, hexapod or individual motors. We use EPICS GUI written in EDM or MEDM [1]. Panels hide the complexity of the software. Very popular way of interfacing MCS-8 is by using SPEC [2]. SPEC is a package for instrument control and data acquisitions widely used for X-ray diffraction at synchrotrons. MCS-8 comes without embedded computer if used with SPEC, since SPEC is usually installed on a remote linux workstation.

This article describes the components of the MCS-8: Turbo PMAC 2, microIOC and various power drives. Second part presents an example of usage of MCS-8 to control hexapod acting as optical element in ADRESS beamline in Swiss Light Source (SLS).

MCS-8 COMPONENTS

Turbo PMAC 2

Turbo PMAC 2 is the heart of MCS-8. It uses the increased speed and memory of the newest generation of digital signal processing (DSP) ICs. It has capability to control up to 32 axes in 16 independent coordinate systems. Turbo PMAC 2 board itself has at most 8 axis interface channels. To actually control more than 8 physical axis users must use either special expansion board or remote interface circuitry on the MACRO ring [3]. The DSP of the base version of turbo PMAC 2 runs at 80 MHz. Serial interface is used to directly interact with PMAC (can be used for manual configuration) using Delta Tau software running on Windows. Ethernet is used for internal communication between PMAC and microIOC.

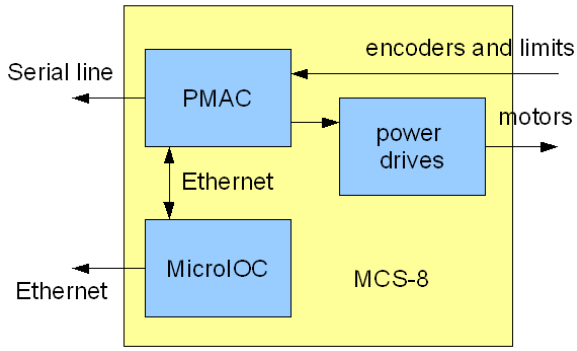


Figure 1: MCS-8 main components

Software on the Turbo PMAC 2 is divided into: setup, motion programs, homing programs, PLC programs and servo and phase algorithms. Setup on Turbo PMAC 2 is performed by means of setting system variables called I variables [3]. There are more than 8000 variables in total and 100 for each motor. I variables are used for settings like maximum velocity, soft limits, PID parameters. Motion programs are programs that are usually used to move the motors. They provide an easy way to specify sequences of coordinated motion of multiple axes and the execution of any calculations that are synchronous with the programmed motions.

Homing programs are needed if the encoder on the motor is relative and the absolute position is lost after a power failure or a hard reset. Basic homing program drives motor to one of the limit switch and then the home reference search sequence is initiated. Other types of homing programs for motors that don't have home reference are also possible (they can use one of the limits switch as a home indicator).

PLC programs are intended for actions and calculations that are asynchronous to the programmed motions. PLC programs repeatedly scan in the fashion of regular programmable logic controllers. They are used for initialization, monitoring of various registers, to perform manifold actions at certain conditions, such as switching off power drives after motion for motors that would otherwise overheat and they can also be used to implement actions which PMAC would perform if communication to the remote controller would fail (e.g. stop all motors).

Servo algorithm implements PID loop for each motor. In this algorithm PMAC calculates required signal for the outputs based on the difference between actual and demanded position. Standard servo loop is shown in figure 2.

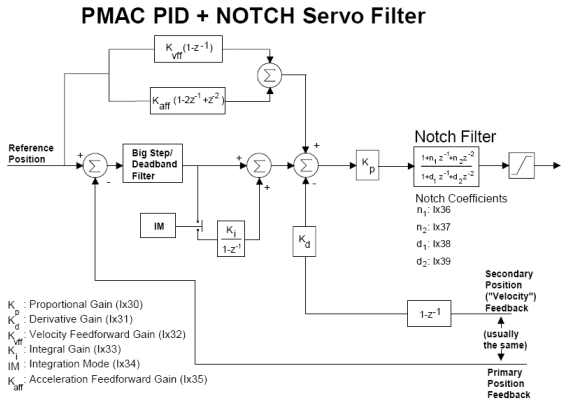


Figure 2: Standard servo loop

Advanced users can write their own servo algorithms. Since these tasks are executed at very high priority they can also be used as a very fast input/output operations or calculations if they are applied on motors that are not used in the system.

MicroIOC

MicroIOC [4] is embedded single board computer (SBC) usually running software for controlling devices. It has standard outputs for mouse, keyboard, VGA monitor, 2 USB plugs and also two Ethernet connections. All these except one internet connection can be accessed from MCS-8 front panel. One of the Ethernet connections is used for internal communication between PMAC and EPICS software. It does not have a hard disk; all software is stored on a compact flash disk (CF) which is loaded at boot time. Most widely used operating system is linux (Debian distribution) but also Windows or RTEMS are possible.

Power drives

The PMAC board provides control for a wide variety of actual drives. Some of the supported are:

- Mounted internally in MCS-8
  - Microstepper Driver – this drive is a PWM Chopper, providing bipolar drive for 2-Phase stepping motors.

- Servo Motor Driver - This is an intelligent PWM servo drive designed to drive brushed and brushless servomotors.

- Other drive options mounted externally to MCS-8:
  - DC brushless motors
  - Intelligent "Picomotor" drives
  - Nanomotors
  - Piezoelectric drives

HEXAPOD APPLICATION

Parallel kinematics manipulators (PKMs) have been rediscovered in the last decade as processor's power finally satisfies computing force required for their control. Their great payload capacity, stiffness and accuracy characteristic as result of their parallel structure make them superior to serial manipulators in many fields.
One of the most accepted PKMs is Stewart-Gough platform based manipulator, also known as hexapod platform. Hexapod consists of two platforms, one fixed on the floor or ceiling and one mobile, connected together via six extensible struts with spherical or other types of joints. That construction gives mobile platform 6-DOF (degrees of freedom). Hexapod movement and control is achieved only through strut lengths changes.

Kinematics

In hexapod motion control it is essential to have capabilities of calculation inverse and forward kinematics. PMAC provides two special buffers, one for forward and one for inverse kinematics. If using kinematics in motion control, no motion program needs to change because PMAC calls kinematics calculations internally. The purpose of forward kinematics is to calculate tool tip position from joint (motor) positions. Inverse kinematics is mathematical inversion of forward kinematics; it calculates joint (motor) positions based from the tool tip positions [5].

In hexapod application user can demand new position of the hexapod only in tool–tip coordinates. PMAC first calculates forward kinematics before a move is initiated. This is done since motors could be individually moved and the starting tool tip position would then not match the real world. Calculated values are used as an input to inverse kinematics calculations. PMAC automatically calls the inverse kinematics for each programmed move typically every 10 milliseconds.

EPICS software

EPICS software runs on a microIOC. Since all major calculations are done inside PMAC EPICS software acts mainly as interface to the user panels. It is used to control and monitor tool tip moves, monitor individual motor status, setup hexapod coordinates systems and scanning functionality. Figure 3 shows an example of EPICS GUI panel. Control in tool tip coordinates is in the right top corner. Lower right part is used to monitor individual motors status (limit switches, failures and velocity).

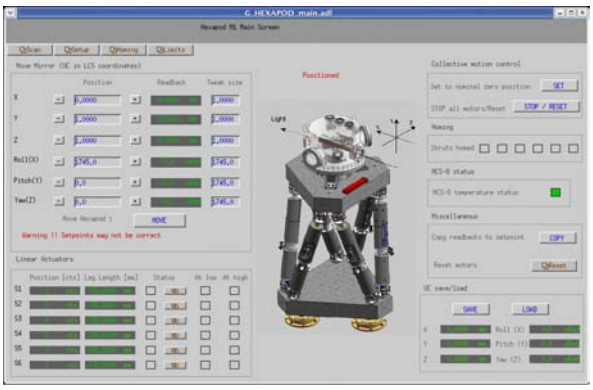


Figure 3: Hexapod user panel

Scan functionality offers user to scan hexapod position in one of the 6 tool tip axis (X, Y, Z, Y, Yaw, Pitch and Roll). User must select start position, end position, number of points in scan and dwell time in each point. For each point the software is able to obtain values from 6 different sensors. At the end of scan graph with acquired data is displayed and peak statistics is calculated.

REFERENCES

[1] www.aps.anl.gov/epics/extensions/medm/index.php http://ics-web1.sns.ornl.gov/edm/

[2] http://www.certif.com/spec.html

[3] "Turbo PMAC – User manual", Delta Tau, October 2004

[4] R. Sabjan et al., "microIOC and EPICS: The Complete Control System Node", PCaPAC 2005

[5] Lung-Wen Tsai, "Robot Analysis"

ETHERNET BASED EMBEDDED SYSTEM FOR FEL DIAGNOSTICS AND CONTROLS

J. Yan, D. Sexton, W. Moore, A. Grippo, and K. Jordan
FEL, Jefferson Lab, Newport News, USA.

Abstract

An Ethernet based embedded system has been developed to upgrade the Beam Viewer and Beam Position Monitor (BPM) systems within the free-electron laser (FEL) project at Jefferson Lab. The embedded microcontroller was mounted on the front-end I/O cards with software packages such as Experimental Physics and Industrial Control System (EPICS) and Real Time Executive for Multiprocessor System (RTEMS) running as an Input/Output Controller (IOC). By cross compiling with the EPICS, the RTEMS kernel, IOC device supports, and databases all of these can be downloaded into the microcontroller. The first version of the BPM electronics based on the embedded controller was built and is currently running in our FEL system. The new version of BPM that will use a Single Board IOC (SBIOC), which integrates with an Field Programming Gate Array (FPGA) and a ColdFire embedded microcontroller, is presently under development. The new system has the features of a low cost IOC, an open source real-time operating system, plug&play-like ease of installation and flexibility, and provides a much more localized solution.

INTRODUCTION

The Free Electron Laser Project at Jefferson Lab is an electron accelerator that provides intense, powerful beams of tunable infrared (IR) laser power. This complicated machine requires numerous control and diagnostic systems, some requiring high levels of precision. Currently these devices, such as beam viewers and BPMs, are controlled, configured and monitored by a central VME bus-based configuration. Within the FEL there are over 100 beam viewers and 35 BPMs that are distributed throughout the IR FEL beam line, which means high cost for cable and maintenance. The UV beam line under construction needs about twenty more beam viewers and twenty BPMs. However, the current configuration is limited, so an upgrade for the beam viewer system and BPM is required for machine expansion and growth.

Currently we have two kinds of configurations for the BPM system. Figure 1(a) shows the BPM control system with a VME crate with a custom designed set of SEE electronics. Each BPM connects with a set of SEE electronics a BPM Can. Figure 1(b) shows the BPM system with both VME and CAMAC crates. The adapter on the VME crate talks with the CAMAC controller. Within the CAMAC crate each board controls one BPM device. If we want to add more BPM devices, we either need a VME crate or a CAMAC, and also a lot of cable. The cost will be very high.

The beam viewer system is also based on the VME bus. The IP packs, both IP-DIO48 and CAN bus adapter, are currently used to control the chassis, which has an electronics board for each slot and one board controlling one beam viewer, with a total of 12 channels per chassis. Expensive and obsolete chassis are required to expand the beam viewer system.

An Ethernet based embedded system was proposed to solve these problems and other control system upgrades within the FEL. Low cost, powerful embedded microcontrollers make it possible to network numerous devices and have stand alone systems.

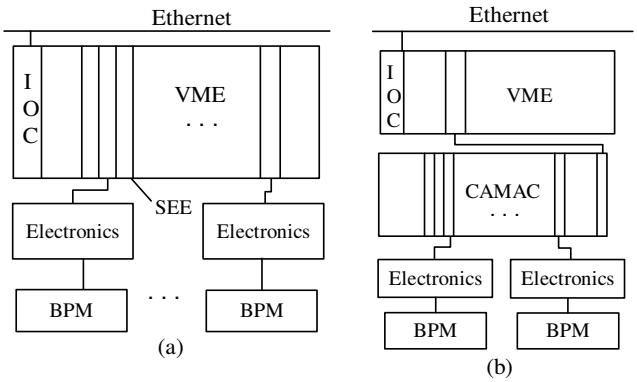


Figure 1 The current BPM control systems

NEW SYSTEM DESIGN

Microcontroller

The embedded microcontroller is the uCdimm Coldfire 5282 module (uC5282), which is a complete "system on a module", including three basic highly integrated functional blocks, core processor, memory and Ethernet. The module provides abundant high speed serial and parallel I/O ports on board when coupled with suitable devices. It can directly control the I/O devices without external support. The board has the following configuration, Motorola MCF5282 controller with 64 MHz Coldfire RISC, 512 KB embedded flash memory for Bootloader and environment storage, 16 MB on-board RAM, enhanced multiply-Accumulate Unit (eMAC) for DSF functionality, 10/100 Ethernet MAC for fast Ethernet support, two RS232 serial UARTs, queued serial peripheral interface (QSPI) with four peripheral chip selects, 8-channel 10-bit queued analog-to-digital converter (QADC), and other multi-purpose input/outputs.

RTEMS and EPICS

RTEMS is a full featured real-time operating system that supports a variety of open API and interface standards. It was developed by OaR Corp, and aims to provide a free real-time operating system for deeply embedded systems with less memory. Its performance is

competitive with close products[1,2]. The system can be tailored to a specific application by choosing an appropriate configuration, where various system components are partitioned into separate modules. RTEMS is designed to be easily portable and consequently supports many CPU architectures, such as m68k, ColdFire, Intel i386, PowerPC, SPARC, AMD A29k, and so on. Some previous work had been done to compare the network and latency performances between RTEMS and vxWorks. It has been proven that the real-time performance of the RTEMS is comparable to that of a commercial system like vxWorks[1,3]. EPICS has been ported to RTEMS as of new 3.14 EPICS release[4]. The embedded ColdFire microprocessor integrated with RTEMS and EPICS allows the microprocessor to be configured as an IOC This provides the advantages of low cost, flexibility and optimal real-time control systems.

BPM

The first version of the BPM stand alone system is divided into two major sub-sections, the RF Front End electronics, which converts 1.497 GHz RF input into DC, and the Digital Section with the ColdFire IOC, which is responsible for data processing and network communication. Figure 2 shows the schematic of the new BPM configuration. The RF signals from four channels, X^+, X^-, Y^+, and Y^-, are converted into DC voltages by an AD8362 and sampled by a 16-bit ADC ADS8364. The Coldfire processor reads four-channels of digital data from the ADS8364 through the digital I/O pins. Figure 3 shows an overall picture of the BPM electronics with the on board Coldfire uC5282 processor.

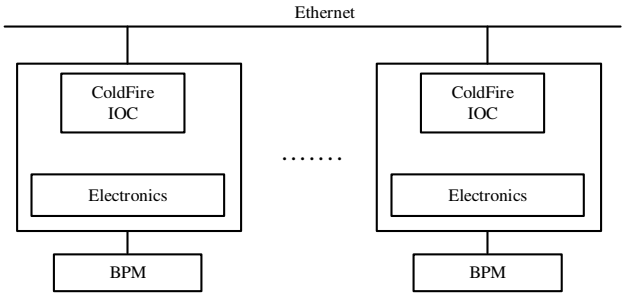


Fig. 2 The Schematic of new BPM control system

The device drivers, device supports, and sequencer were developed under RTEMS/EPICS tools to process the data calculations and Spline interpolaton calibrations for the beam position. The embedded Coldfire IOC directly communicates with the EPICS controls system through the on board Ethernet interface. Each of the embedded IOCs has the functionality of remote programming and remote reboots. This allows for quick changes to be uploaded into the IOC, without having to remove the device from the field. Some of new BPMs have been running within the FEL system for more than one year. They have proven to be very reliable, easily maintained. The control screens of the new BPM are compatible with the previous BPM system. However, one limitation of this prototype is the throughput from the ADC to the Coldfire uC5282 processor, as this is limited to about 100 Hz. This limits the resolution that these electronics are capable of achieving. A new version of BPM based on the SBIOC, which integrated the FPGA and the Coldfire uC5282 processor into one device, is under development. This configuration will solve the limitation and dramatically increase the throughput. The configuration of the SBIOC board will be described in the next section.


Fig. 3 The overall picture of the BPM electronics

SINGLE BOARD IOC (SBIOC)

Because the first version of embedded BPM does not have a high sample rate to provide high resolution, a solution to increase the sample rate is currently being tested. The method of implementing an FPGA to speed-up the digital sample rate was applied. FPGAs are very attractive for implementation into digital designs, because they have high component counts and provide enormous design flexibility with relatively low cost. By integrating the FPGA and the Coldfire uC5282 processor, the SBIOC was designed.

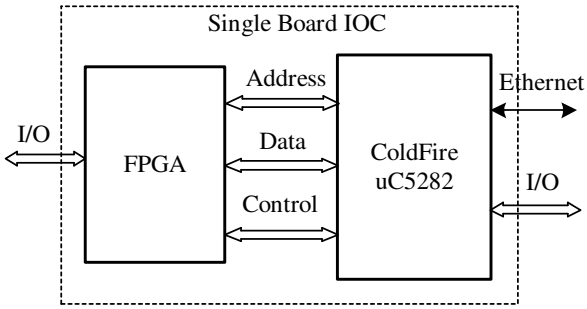


Fig. 4 The schematic of the Single Boardd IOC

Figure 4 shows the schematic of the SBIOC. The Altera Cyclone II EP2C8Q208 FPGA is chosen, because it provides abundant I/O pins for device control and the connection with the Coldfire uC5282 processor. The interface between the FPGA and the Coldfire uC5282 processor includes a 24-bit address bus, a 16-bit data bus, and a control bus. The clock of the FPGA can either be driven by the Coldfire Processor clock or an external crystal oscillator. The SBIOC has two 40-pin headers and that allow it to drop in a carrier board as a module. Since the new board is pin-pin compatible with some existing designs, it can be upgraded and replaced without have to redo the whole carrier card. The two 40-pin sets have 32 bits I/O pins from the FGPA, 8 bits of general purpose I/O

pins, several other generic digital bits, Ethernet and RS-232 communication pins, as well as 5V or 3.3 V power lines. Figure 5 shows the pictures of the SBIOC prototype. a) front view with the FPGA and without Coldfire uC5282 processor, b) with the ColdFire uC5282 processor plugged on the top of the FPGA

(a)

(b)

Fig. 5 The pictures of Single Board IOC, (a) without the uC5282, (b) with the uC5282

The functional logic of the FPGA was designed with the Quartus II software tool. The typical functional modules of the FPGA consist of a Coldfire-FPGA bridge, a control module, an interface logic of FPGA and I/O devices, and a memory block. The Coldfire-FPGA bridge is the communication protocol between the FPGA and the Coldfire uC5282 processor, which controls the address bus, data bus and control bus. Through the bridge the Coldfire uC5282 processor reads and writes the FPGA as a peripheral device. The control module coordinates other modules. The interface logic module interacts with I/O devices while the memory block is RAM in the FPGA. Once complied, all the code can be stored in an EPROM device that connects with the FPGA.

The SBIOC is used for the new version BPM. The FPGA collects the data from the ADC as fast as the ADC maximum throughput. For example, AD7655 has sample rate of 1 MHz, ADS8364 has 250 KHz for each channel. The FPGA will do some averaging to reduce the noise. The data that are sampled from the ADC will be stored in the memory block in the FPGA. The Coldfire processor reads the data from the memory and executes all the calculations and calibrations. This BPM will have self triggering capabilities, fast throughput for averaging, and better position resolution. By using the FPGA, it is possible that we can provide some beam diagnostics that we couldn't with the current system. Some other control sub-systems, such as beam viewer crate, GC chassis, Charge/Dump chassis, and vacuum crate, will be upgraded by using the SBIOC. All these systems will have same hardware platform, the only difference is the function of software. That will save a lot of cost and should be easy to maintain.

CONCLUSIONS

The prototype of the SBIOC has been designed and built. Some function modules in the FPGA were programmed and tested. The Coldfire-FPGA bridge was programmed and the uC5282 processor could access the FPGA through the address bus and data bus. The interface module in the FPGA could sample the data from the ADC. The SBIOC was tested with the BPM electronics, and we will design it into the next revison of the BPM electronics. The new Beam Viewer card is under development. More work of the SBIOC has to be done to improve the performance.

The Ethernet based SBIOC integrating the FPGA with the ColdFire uC5282 processor provides a new configuration for the upgrade of our FEL control and diagnostic systems. By using the open source RTEMS and EPICS real-time control system, there will be no license hassles and less cost. By moving the IOCs to the frond end, we cut the cost of expensive cables. The SBIOC will make the system more powerful and flexible, and it will improve the performance of FEL. This configuration can be used for the next generation FELs.

ACKNOWLEDGEMENTS

Thank Eric Norum in Argonne National Lab for providing RTEMS tools and the step-by-step installation instructions. Thank Stephen Dutton for electronics board making and Trent Allison in ESICS group of Jefferson Lab for the help of Altera FPGA programming. Thank the Operations and Commissioning team of the FEL for the support and technical advice. Also a special thanks to Pavel Evtushenko for his expert advice and explanations in BPM operations and calculations.

This work is supported by the Office of Naval Research, the Joint Technology Office, The Commonwealth of Virginia, the Army Night Vision Laboratory, the Air Force Research Laboratory, and by DOE Contract DE-AC05-84ER40150.

REFERENCES

[1] Till Straumann, "Open Source Real-Time Operating Systems Overview", cs,os/0111035 (2001) WEB1001

[2] http://www.rtems.com/features.html

[3] S. K. Feng, and etc., "EPICS/RTEMS/MVME5500 For Real-Time Controls at NSLS", 10th ICALEPCS Int. Conf. on Accelerator & Large Expt. Physics Control Systems. Genava, 10-14 Oct. 2005, TU4A. 1-50 (2005)

[4] http://www.aps.anl.gov/epics/base/RTEMS.php

[5] D. Sexton, etc., "Development of BPM Electronics at the JLAB FEL", 12th Beam Instrumentation Workshop, Fermi Lab, Batavia, IL, May 1st-4th, 2006.

MICROIOC: PC AND CONTROL SYSTEM LONGEVITY*

A.Podborsek#, D. Golob, A. Hasanovic, I. Kriznar, R. Sabjan, M. Plesko, Cosylab, Slovenia

Abstract

Products, taking part in control systems (CS), belong to specific area of system solutions. From a high-level point of view a CS can be split onto a main control part, managing all important high-level CS aspects, and peripheral control part, managing distributed end-point devices.

These two parts feature different requirements because of the environment and the role they are used in. Nevertheless, both parts should be seamlessly integrated in order to lay foundations for reliable and modular CS.

Main control part is normally situated in human and electronics friendly environment and aimed at high-level engineering control. On the contrary, a peripheral control part is usually used in industrial environment with no direct human interaction. Because of a diversity of end-point devices, flexible and user customizable interfaces are required for peripheral control. Long years of operation are required until end-point devices (data acquisition, actuators, etc.) are being replaced or additional functionality is added. Reliability and future modularity is therefore of crucial importance.

When it comes to the length of service of the entire CS, it is expected, the same remote control solution would provide adequate functionality through the whole lifespan of system where it is used. Designers at Cosylab – as a provider of CS solutions – we all strive for longevity of our solutions. In other words, it is the length of service we are after. This paper is focused on issues and solutions for the peripheral part of the CS.

INTRODUCTION

Centralized control systems (CS) are considered to be split into two important subparts – into main control part and peripheral control part (Figure 1). Whilst integrating CS solution, both parts must be carefully weighted for the role they take over. CS integrators are targeting robust, modular and flexible solutions.

In the main control part of the system it is usually required that wide variety of system aspects is handled and usually some form of data processing must be applied. It is important to have a system with adequate data processing power and flexible and user-friendly way of managing of all complex aspects of system under control. Typically such a system is managed by control-system engineer in a human friendly environment of central control room. It is therefore a typical domain for personal computers (PC) to provide this control flexibility and processing power.

At peripheral control quite different conditions and requirements apply: demanding industrial environment, large number of end nodes, wide range of required interfaces towards controlled devices, and different system roles. Nevertheless, complex general-purpose data processing is usually not required as all potential complex processing is handled over to the main control part, which has access to all necessary system-level information. If a demand for computer-intensive data processing exists, custom embedded solutions are used, like for example in motion control applications [5].

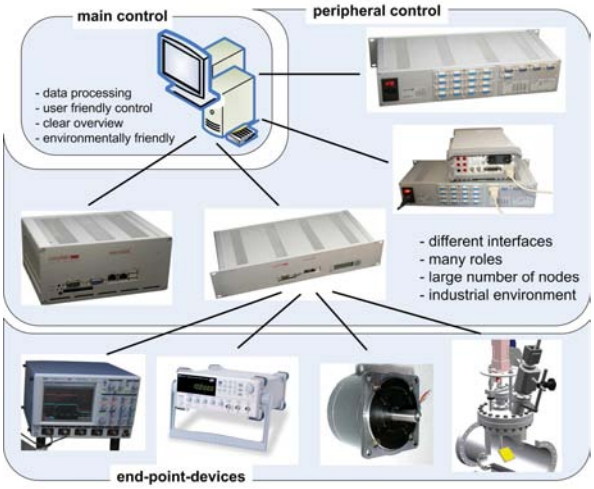


Figure 1 – control system

Based on the above, it can be concluded that peripheral control is successfully handled only if peripheral control is systematically addressed. A stress must be put onto flexibility of connectivity, modularity, reliability and durability. Especially if peripheral control devices are to be integrated in complex scientific facilities, it is expected they will play their role flawlessly.

Solutions on the market

Many different solutions exist on the market. One, interested in integrating a CS solution, should take into account price-effectiveness, size, modularity, scalability, robustness, and integration friendliness/readiness. It must be pointed out that there is no such thing as the best peripheral-control product, at least not for every CS scenario. All before mentioned criteria must be carefully weighted.

When talking about price-effectiveness, an overhead price, added to the device, requiring remote control, must be considered and cost for adding support for additional device of the same/similar type. This is also the issue of modularity and scalability, i.e. the possibility to add additional device(s) of the same or different type.

In case multiple devices must be controlled, physical dimensions could determine the feasibility of the proposed solution.

In the system development phase, possibility and support for system-integration are of great importance, determining whether the implementation deadlines can be met or even if the proposed solution can be finalized.

Some of the most commonly used solutions for peripheral control are:

- VME (Versa Module Eurocard) crate based solutions or VXI (VMEbus eXtensions for Instrumentation) [6];
- personal computer (PC) based solutions;
- embedded PC solutions with customized input-output controller.

VME crate solutions are generally reliable and modular. Additional functionality can be simply added by the use of broad range of available extension cards. A special care should be taken if only a small portion of a VME crate is required. In such cases VME-crate solutions could lead to very poor price-effective solutions. Additional issue could be density of cards connectors, as this often requires the use of special transition boards.

In some special circumstances a PC could be used for remote control. It is ubiquitous, features high computing power and highly flexible SW support, but the choice of peripheral control cards is not so rich as with VME crate solutions. Industrial environment, moving parts (power supply ventilators, hard disks, etc.) and its size could represent issues when durable and robust peripheral CS solution is sought.

Embedded PC solutions can be considered as an intersection of both previously presented peripheral control devices. Embedded single board computers (SBC) represent a flexible HW platform, offering common PC functionality. The use of standardized operating systems (e.g. Linux or Windows Embedded) and extension buses (e.g. PC/104) both contribute to flexible platform, ready to be customized for nearly every peripheral CS role. Mass production of embedded SBC platform and variety of extension cards guarantee the possibility to choose just the right-configuration for the task. Because they are optimized for size and often also for power consumption, they generally use no moving part (neither for data storage nor for cooling), which lays a good starting point for use in demanding industrial environment. With the use of industrial-grade components, these systems are targeting high MTBF (mean time between failures) ratios.

MICROIOC

microIOC is an example of embedded PC-based peripheral device, especially built for use in remote control applications [1]. Its modular design and support for specific customization scenarios makes it a perfect turn-key solution for the control of any kind of end-point devices. Essentially it is a bridge that integrates the devices with the rest of the CS. The devices are connected to the microIOC, while the microIOC itself is available to the control system via the local area network [2].

Interfacing and installation effort is minimized, as connector customization and support for integration into higher-level central CS are also included in application specific customization.

Its cautious design and rigorous testing provide grounds for playing its role flawlessly. microIOC was especially designed with a length-of-service in mind. A lot of effort was put into designing a system with all components being highly reliable. The core of the unit is composed of SBC, compact flash (CF) memory module, and power supply, all of them being carefully tested.

Support for various end-point devices and integration into higher-level CS is provided through highly reliable and industry-proved Linux operating system and preinstalled support for direct integration into standard CS solutions.

Components are enclosed in a compact and robust aluminum casing, available in various form factors. Usage of external cables is minimized to minimum: mains supply cable, control system connection and connections towards controlled devices. No other distribution cables are required.

Figure 2 – examples of microIOC units

It offers a wide range of communication interfaces (RS232, RS422, RS485, GPIB, Ethernet, USB, etc.), which can be used to provide communication means for a variety of devices, for which remote control is required.

Reliable

When reliability comes to question, both hardware and software are equally important. microIOC is designed using reliable and long-life industrial components. All modules are carefully chosen in the way no maintenance is required. As such it is perfectly suited for integration into demanding industrial environment. No moving parts are used in preferred hardware configuration. Even a fan and a disk are left out, as this provides a system that does not produce any noise and is resistant to vibrations. In default configuration ultra low-power processor is used, supplied with a small and reliable power supply.

Wherever possible, high-quality standard components are used. These components feature higher reliability factor because of their wide usage in larger number of different applications; consequently their long-term operating is tested much better.

Modular and configurable

microIOC is bottom-up designed to be highly customizable platform. Starting with 3 standard types of enclosure a variety of mounting requirements can be met. In basic version one SBC can be used. If computational power and/or interfacing requirements are greater, additional SBC is used. If required, application specific circuitry can be developed to precisely meet application requirements. This forms a very flexible base, which can be modularly extended by the use of standard PC/104 extension cards. Various extension cards and additional modules can be integrated into microIOC unit to accommodate all remote control requirements.

When it comes to connecting controlled end-point devices, microIOC unit is equally customizable as with physical dimensions and functionality. No special transition boards are required that take-up additional rack space. Mains connection and switch occupy very little portion of the microIOC back panel, the remaining of the space can be used to mount application customized back-panel connectors for direct connection of devices.

Software

When reliability, flexibility and applicability of embedded application are at question, Linux is operating system (OS) of choice. Its kernel can be adjusted to include just the required functionality, without system overhead. Our policy is to use the kernels that are already widely used, supported and tested [3].

microIOC uses CF memory card as non-volatile memory to store Linux kernel image and required file system. microIOC is distributed with a SW configuration matching HW configuration. That means that all required drivers are provided and SW properly configured.

Open source OS gives a user opportunity to precisely implement any peculiar design requirement. Furthermore, it relieves the user of relying on specific commercial provider if additional requirements show up at some later time. It is fairly to assume that future peripheral CS requirements will not become so drastically computing intensive, therefore only flexible and modular base is required and we can target the length-of-service.

For integrating into the main control system, microIOC is provided with a support for EPICS (Experimental Physics and Industrial Control System) control software [4]. Device specific code and low-level communication drivers communicate via asynDriver.

Software support for user customization is provided through the use of development environment. It is designed to provide support for integrating custom functionality and/or new device support.

TESTING

All components comprising the microIOC are tested for compatibility and long operation under different conditions, including abusive usage. Series of tests have been made: long term operation test, memory test (including three month test for memory leaks), temperature test, CF test with random power off/on and CPU overloading test. Any microIOC unit, leaving our production, is tested as well to meet applicable quality standards.

microIOC components test

Besides requiring industrial robustness and high MTBF times, a series of test are made to ensure that a reliable system can be built. All components that are used in microIOC undergo a series of rigorous testing procedures, including: automatic optical inspection (AOI), function testing, environment testing (temperature / humidity / vibration / drop test), Static and Dynamic burn-in test. Even though we require that all our components have all the tests already made by the manufacturer, we still repeat some of them when a system is fully assembled in operational.

Compact flash random test

Based on our experiences with several different types of Compact Flash (CF) cards, we came to the conclusion that some of them feature degraded performance. In order to ensure reliability of a produced system, several tests had to be made. The most important of them was a test, where a power supply was put on and off repeatedly in a random manner. Different models of CF cards were tested at largest obtainable read/write cycle. The results of the test were unexpected. Some of the world-class brand-name CF cards models were found that were featuring unsatisfying performance. There were also examples of CF cards that either became defective or its data-system became corrupted and unusable.

While testing CF cards one of the tests was also to utilize 15,000 of power-up/boot procedures. This is a very strict demand and if CF card does not meet this demand, it is discarded from our future use.

Temperature stress test

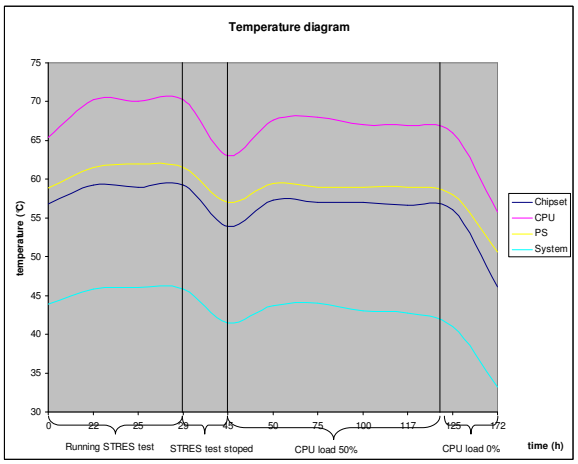


Figure 3 - temperature stress test of the microIOC

The purpose of temperature stress test was to determine the stability of the system. If specific application requires the use of multiple additional boards, they all contribute to heat that has to be dissipated by means of convectional cooling. If unit is to be used rack mounted, higher environmental temperature must be taken into account. Several temperature tests were performed to observe and demonstrate microIOC would perform its task flawlessly, even if used in demanding industrial environment (Figure 3).

CONCLUSION

Peripheral control of end-point devices belong to special area of CS solutions. Varieties of devices to be controlled require peripheral CS solutions to be highly flexible and customizable. Demanding environmental conditions require industrial-grade reliability. Designed with careful selection of components and additionally rigorously tested, microIOC represents a turn-key solution when such remote control is required. Both HW and SW support is provided within the box. Installation is quick and trouble free. Presented design procedure and integration approach set a firm ground for quality and length of service, i.e. longevity of CS solution.

REFERENCES

[1] M. Plesko, "New developments at Cosylab", PCaPAC '05, Hayaman, 2005

[2] R. Sabjan, "microIOC and Epics: the complete control system node", PCaPAC '05, Hayaman, 2005,

[3] http://www.debian.org

[4] Experimental Physics and Industrial Control System: http://www.aps.anl.gov/epics/

[5] Delta Tau motion controller solutions: http://www.deltatau.com

[6] VME and VXI specifications: http://vme-vxi.globalspec.com/

ETHERNET-BASED FIELDBUS FUNCTIONALITY FOR NEUTRON SCATTERING EXPERIMENTS WITH PROFINET IO

Harald Kleines, Sebastian Detert, Frank Suxdorf, Matthias Drochner,
ZEL, Forschungszentrum Jülich, Germany

Abstract

Recently the complex PROFINET CBA (Component Based Automation) has been supplemented by the simplified PROFINET IO standard. PROFINET CBA defines a vendor-independent engineering model covering plant-wide automation and uses a component/container approach. Contrary to that, PROFINET IO is targeted at decentral periphery scenarios. Analogous to PROFIBUS DP it is based on a modular device model. Functionally, PROFINET IO can be considered as an Ethernet-based fieldbus.

PROFINET IO is evaluated in Forschungszentrum Jülich for future application in neutron scattering experiments. The paper introduces the PROFINET IO technology and presents the plans for the positioning of PROFINET IO in the control system architecture of neutron scattering experiments.

INTRODUCTION

In order to further strengthen its neutron research, Forschungszentrum Jülich founded the JCNS (Jülich Center of Neutron Science) on its own campus with branch labs at the ILL in Grenoble, at the SNS in Oak Ridge and at the FRM-II (Forschungsreaktor München II). FRM-II is a new high flux neutron source operated by the Technical University of Munich in Garching near Munich. JCNS will operate 7 neutron scattering experiments at the FRMII, which are under construction now, partly based on experiments that have been operated at the research reactor FRJ-2 in Jülich before its shutdown in May 2006. These instruments will get new control and data acquisition systems, since software and electronics typically are older than 10 years.

ZEL (Zentralinstitut für Elektronik), the central electronics facility of Forschungszentrum Jülich responsible for the design and implementation of all new control and data acquisition systems for neutron instruments in Jülich has started a close cooperation with the instrumentation group at the FRM-II. Together both defined a common framework for all new control and data acquisition systems of neutron instruments in Garching, the so-called "Jülich-Munich Standard", which is followed by most instruments at the FRM-II. A guiding principle for definition of the framework was to minimize the development efforts and to acquire as much from the market as possible.

Slow control in neutron scattering experiments is related to the accurate movement of a diverse range of mechanical parts, to pressure or temperature control and safety instrumentation. Because ZEL introduced industrial control equipment already in the 80s to experiment instrumentation, a key component of the framework is the consequent use of industrial technologies like PLCs, fieldbus systems or decentral periphery in the front end. Main motivations are:

- low prices induced by mass market,
- inherent robustness
- long term availability and support from manufacturer
- powerful development tools

Since Siemens is the dominating supplier for PLCs in Europe, the front-end systems being build by ZEL are based on Siemens products, especially S7-300 PLCs and ET200S decentral periphery connected via PROFIBUS DP.

PROFIBUS DP has been designed for the connection of decentral periphery systems to a central PLC. But ZEL uses it also for the communication between PLCs or other process devices and the supervisory computer, since PROFIBUS DP, which now is the world's leading fieldbus, has become a de facto standard with products available from many companies [1]. It is especially well supported by Siemens PLCs used in Jülich and provides high performance and a simple and efficient communication model.

On the other hand this approach requires the existence of a PROFIBUS DP communication controller in supervisory computers. There are several controllers on the market, but Linux and CompactPCI, which are used in Jülich, are not well supported. So ZEL had to develop its own PROFIBUS DP communication controller, which requires modifications of the device driver with most Linux version changes.

The now omni-present and cheap Ethernet in combination with the TCP/IP stack could be a possible alternative solving the above trade-off. But since Ethernet and TCP/IP have been designed for an office environment, appropriate application protocols for device communication have been missing in the past. Because of its missing real time features Ethernet has also been considered inappropriate for the factory floor.

This changed during the last years, since several initiatives have been started to establish Ethernet as a fieldbus either by hardware support or by optimized higher level protocols. Examples are [2]

- Ethernet/IP (supported by PLC-manufacturer Allen Bradley),
- Modus/TCP (supported by PLC-manufacturer Schneider),
- Powerlink,

Paper submitted October 20, 2006

- EtherCAT (supported by company Beckhoff) and
- PROFINET (supported by PLC-manufacturer Siemens).

Having several competing protocols leads of course to a market fragmentation and only a few will gain major relevance on the market. The other protocols will remain proprietary solutions of the supporting PLC-manufacturers. For the Neutron scattering experiments of JCNS PROFINET is of major relevance, since it is the only protocol supported by Siemens S7 PLCs. Also it can be expected, that PROFINET gets a similar significance as PROFIBUS got on the fieldbus market, since Siemens is the world-leading manufacturer of PLCs.

Originally only the version CBA (Component Based Automation) has been defined, which is targeted at the communication between intelligent automation devices or computers. PROFINET CBA, internationally standardized by IEC 61784-1[3], defines a vendor-independent engineering model covering plant-wide automation and uses a component/container approach. Recently, the complex PROFINET CBA has been supplemented by the much simpler PROFINET IO standard. Contrary to CBA, PROFINET IO is targeted at decentral periphery scenarios. Analogous to PROFIBUS DP, it is based on a modular device model and relies on the cyclic exchange of messages between device and supervisory system, typically a PLC. Also PROFNET IO has been defined by the PROFIBUS User Organization [4] and it will be included in the international standard IEC61874-2.

OVERVIEW OF PROFINET CBA

The PROFINET CBA concept supports the structuring of an automation system into autonomous subsystems called technological modules. Technological modules comprise all mechanics, electronics and software to perform a specific task. PROFINET CBA uses an object oriented approach by modelling each technological module as a component. The external behaviour of a component is described by interface variables. Components can be reused for different automation systems.

In an engineering phase a specific automation system is formed by defining its components and the communication connections between their input and output variables. Component creation is done by configuring and programming a device in a conventional way with a vendor-specific tool, modelling it as PROFINET CBA component and producing a vendor-independent XML description of this component.

The communication between components is defined by a vendor-independent PROFINET CBA connection editor on an engineering station that allows the import of the XML descriptions and definition of connections between component variables. The connection editor supports download of component and connection configurations to the PROFINET CBA devices.

During runtime all components autonomously exchange data according to their predefined connections without application interaction. Status of devices and connections can be monitored by the engineering station via the diagnostic interface of PROFINET CBA.

Each PROFINET CBA device is modelled as a standardized collection of COM objects. Some of these objects exist only during runtime. As a consequence, all communication for engineering, runtime and diagnostics is based on the DCOM middleware on top of TCP/IP and Ethernet. For real time data an additional Soft Real Time (SRT) stack has been defined. Existing PROFIBUS DP installations can be integrated in a PROFINET CBA system via Proxies.

OVERVIEW OF PROFINET IO

PROFINET IO model

Similar to PROFIBUS DP, PROFINET IO is targeted at application scenarios, where a central station communicates with decentral field devices. As a consequence, each station in a PROFIBUS IO system can take one of the following roles:

- IO controller: The IO controller represents an intelligent central station, like a PLC. It is responsible for the configuration or parameterization of its associated devices and is the source of the output data and the destination of the input data.
- IO device: The IO device represents a field device, like an analogue input unit. It cyclically transmits collected process data to the IO controller and vice versa. It also provides diagnostic or alarm information to the IO controller.
- IO supervisor: The IO supervisor represents an engineering station for programming, configuration or diagnostics, like a PLC programming tool.

PROFINET I/O is based on a consistent model of the IO device structure and capabilities. An IO device may be modular and is composed out of one ore more slots, which may have subslots. Each slot or subslot represents an IO module and has a fixed number of input and output bits. The input data of the IO device is the sequence of the all inputs of slots and subslots, according to their position in the device. The same holds for the output data. Slot 0 and subslot 0 do not represent IO modules and have no IO data. Slot 0 is used to address the IO device and subslot 0 is used to address its corresponding slot. Also all diagnostic or alarm data reference slots or subslots.

Similar to PROFIBUS DP, each device type is defined by a so-called GSD-file in XML format, which describes the device capabilities. The GSD-file defines all possible modules and contains parameter values and descriptions. A vendor independent configuration tool, like a PLC programming tool, reads the GSD-file and allows the definition of the modular structure and the parameterization of each device in a PROFINET IO system. This definition is downloaded to the IO controller, which uses this information to configure and parameterize all its associated IO devices during runtime before entering the cyclic data exchange mode.

Also for PROFINET IO proxies are available on the market, which allow the integration of existing PROIBUS DP installations.

PROFINET IO operation

The IO controller initiates the system start-up as a sequence of several phases, based on the configuration defined with the engineering tool and the information extracted from the GSD-files.

- Each IO devices is checked and the IP-address is assigned with the Discovery and Configuration Protocol (DCP).
- An Application Relation and subordinate Communication Relations to all IO devices are formed with the Context Management services.
- With the acyclic record data services all IO devices and their submodules are configured and parameterized.
- After successful configuration and parameterization an IO device enters the cyclic data exchange mode, where process data and high priority alarms are exchanged cyclically with the RT communication services [5].

PROFINET IO protocol

PROFINET IO distinguishes between non real time communication, real time (RT) communication and isochronous real time (IRT) communication. RT and IRT communication are only used for the cyclic data exchange mode and the DCP protocol. All other communication is non real time and is defined as an application protocol on top of TCP/IP.

In order to guarantee a cycle time in the order of a few milliseconds with moderate jitter in soft real time scenarios, the RT communication bypasses the TCP/IP stack and directly uses the MAC layer of Ethernet. RT frames are identified by the value 0x8892 in the ethertype field.

For hard real time scenarios, e.g. for motion control applications, the IRT communication is used, which guarantees a cycle time with jitter below 1µs. IRT employ the "Precision Transparent Clock Protocol" (PTCP) according to IEC61158 for clock synchronization and requires hardware support by ASICs.

PROFINET IO FOR NEUTRON SCATTERING AT JCNS

PROFINET IO products are available on the market from several vendors. It is especially well supported by Siemens, which provides PROFINET IO communication modules for PLCs as well as PLC CPUs with integrated PROFIBUS IO. Also decentral periphery systems (ET200S, ET200pro) are available for PROFINET IO. Out of the perspective of a PLC programmer, IO devices can be access like local PLC IOs with the statements "load" and "transfer" in the programming language Step 7 STL (statement list), exactly as with PROFIBUS DP. Also for the transfer of large data blocks and for the access to diagnostics or alarms the same or similar functions as for PROFIBUS DP are used. With regard to performance, we measured the time 6 ms to transfer 1 Byte from a PLC CPU 315-2 PN to a ET200S digital output, which is almost the same as for PROFIBUS DP. This mostly due to the low speed of the ET200s processing of output data, since we observed a reaction time on Ethernet of about 0.3 ms.

Since PROFINET IO is much simpler than PROFINET CBA and because of its similarities to PROFIBUS DP, we decided to use PROFINET IO in future for the communication between supervisory computers and subordinate systems, especially PLCs. Communication with a S7-300 PLC as an IO device can be done with recent version of the communication module CP 343-1 Lean, but PROFINET IO controller functionality under Linux is not yet available on the market. So we started to implement a subset of PROFINET IO consisting out of DCP, context management, the acyclic record data services read and write, and the RT communication for cyclic data exchange. As a development environment we set up a test system with a working controller/device-scenario (PLC CPU 315-2 PN/DO as IO controller, IO devices ET200S with interface module IM 151-3PN and a S7-300 PLC with module CP343-1 Lean) and Wireshark as a network monitor, as indicated in Fig. 1.

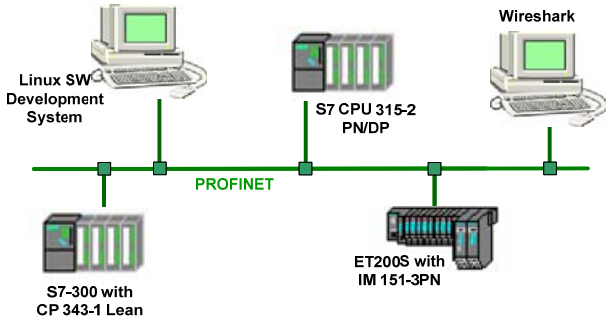


Figure 1: Test system for the SW developments

REFERENCES

[1] H. Kleines et al., Implementation of the Control and Data Acquisition Systems for Neutron Scattering Experiments at the New "Jülich Center for Neutron Science" According to the "Jülich-Munich Standard", ICALEPCS'05, Geneva, Switzerland, 2005.

[2] P. Neumann, A. Pöschmann, "Ethernet-based Real-Time communication with PROFINET IO", WSEAS Transactions on Communications, Issue 5, Volume 4, May 2005.

[3] IEC 61784-1, "Digital data communications for measurement and control – Part 1: Profile sets for continuous and discrete manufacturing relative to fieldbus used in industrial control systems", 2003.

[4] PROFIBUS International, "Specification PROFINET IO Version 2.0", 2005, http://www.profibus.com.

[5] P. Ferrari, A. Flammini, S. Vitturi, "Performance analysis of PROFINET networks", Computer Standards and Interfaces 28, 2006.

EPICS SCA CLIENTS ON THE .NET X64 PLATFORM*

C. Timossi[1] and H. Nishimura[2], LBNL, Berkeley, CA 94720, U.S.A.

Abstract

We have developed a .NET assembly, which we call SCA.NET, which we have been using for building EPICS [1] based control room applications at the Advanced Light Source (ALS)[2]. In this paper we report on our experiences building a 64-bit version of SCA.NET and the underlying channel access libraries for Windows XP x64 (using a dual core AMD Athlon CPU). We also report on our progress in building new accelerator control applications for this environment.

SIMPLE CHANNEL ACCESS AT ALS

Simple Channel Access (SCA)[3] is a library that provides a simplified API for developing Channel Access (CA) clients. SCA was developed at LBNL and has been in heavy use for both accelerator and beamline controls. Since SCA's most common use has been on Windows platforms, we originally packaged it as an ActiveX Control called SCA.COM[4]. This control is easily called by any Windows client supporting Active X (e.g. Labview, Visual Studio).

Although ActiveX controls can be accessed from .NET assemblies we believed a more seamless integration was important for two reasons. First, .NET will be the standard development framework for Windows in the future--on Vista, it will be the only development framework. Second, it looked like a relatively simple task to re-package the ActiveX control as a .NET assembly. In fact, as often happens during a re-write, we found many optimizations that resulted in better performance than the previous component, in the context of the 32-bit version of SCA.NET[5].

MIGRATION TO 64-BIT

Need to Support 64-bit

PCs with 64-bit processors and operating systems, such as Windows XP x64, are finally becoming widely available. Scientific applications, which are accelerator tracking programs for us [6], are already taking advantage of the larger address space and faster execution speed.

On the other hand, typical machine control applications have little need for these advantages as 32-bit has been mostly sufficient.

It is convenient, however, when building model based 64bit control applications, to have access to 64 bit versions of controls libraries to avoid mixing 32 bit and 64 bit libraries.

*Work supported by the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

[1] CATimossi@lbl.gov. [2] H_Nishimura@lbl.gov

Building CA for Windows XP x64

Building CA on x64 bit platforms has been done previously. Even so, some help from the author [7] of the CA software was needed to add a macro to CA base code for the AMD architecture on Windows. Further, although the EPICS build system is both powerful and flexible, we decided to build the libraries in the Visual Studio environment to take advantage of its rich debugging tools.

Finally, a patch previously posted by AMD, which synchronizes the core time stamp counters, needed to be installed [8].

The 2 DLLs built in this fashion (Com.dll and Ca.dll) are categorized by .NET as *unmanaged* because they were built for the win32 environment, not for .NET.

SCA.NET

Unlike the above DLLs, ALS.dll, which implements SCA.NET, is built as a .NET assembly. When .NET assemblies call routines in unmanaged libraries, they do so through a special interface called *Platform Invoke (P/Invoke)*. C# has syntax for P/Invoke that uses the *DllImport* keyword, for example, as shown below:

```
[DllImport("ca.dll")]
unsafe public static extern
   char * ca_message(uint ca_status);
```

.NET considers any code that manipulates pointers as *unsafe*. The C# compiler will generate an error unless code using pointers are labelled with the *unsafe* keyword.

The CA routines used by SCA.NET are similarly wrapped.

It's worth noting that although the CA libraries are separately compiled for both 64-bit (x64) and 32-bit (x86) versions, ALS.dll is a single binary of the .NET assembly built with the Visual Studio build option of "Any CPU". The OS is responsible for loading either the x64 or x86 versions of the DLLs that ALS.dll needs.

32-bit Programs on 64-bit Windows

Although both 32-bit and 64-bit programs can run on 64-bit Windows, 32-bit programs must run in a compatibility layer called *WOW64*. This layer wraps the application in its own 32-bit environment from which it can only call directly into 32-bit DLLs. The resulting overhead from this layer is architecture dependent--AMD processors can execute 32-bit code directly whereas Intel processors have to emulate 32-bit instructions. A 32-bit process may also use inter-process communications (IPC) when calling into a 64-bit library. Interface options are: pipes, messages, signals, ActiveX/COM out-of-process servers, and networking APIs.

Assigning the CPU as a Build Option

When building SCA.NET, it's most convenient to build and deploy separately for x64 and x86. So first the sources for CA and ALS.dll are compiled as native x64 libraries then an installer project is built. This process is repeated for the 32-bit version. Basically, the installer places the output binaries in either "Program Files\LBNL" for x64 binaries or in "Program Files (x86)\LBNL" for 32 bit binaries.

EXAMPLE

When we program SCA/NET client programs that are portable on x86 and x64 platforms, we must ensure that all the libraries are also portable if they are by 3rd parties. We currently use open-source libraries: SourceGrid [9] for string grid and ZedGraph[10] for chart. Both are managed code in C# and portable.

Fig.1 is an example program that reads and displays all the ALS storage ring magnet EPICS channels (287 magnets and 1669 channels) at 1 Hz by using 13 SourceGrid controls on WinForm.

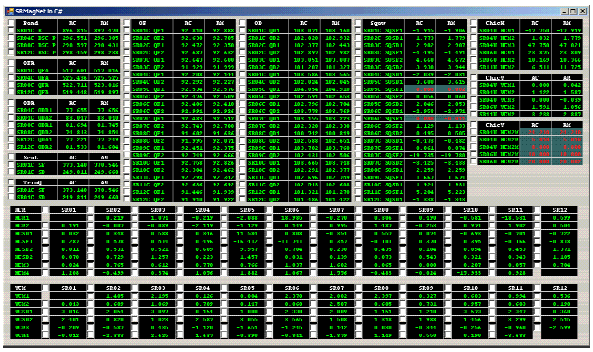


Fig.1. SCA.NET client example

This program runs both on x86 and x64 Windows without rebuilding.

We have also created EPICS database client programs that use ADO.NET 2.0 to access static EPICS database record information such as process variable names from a MySQL[11] database. ADO.NET also allows saving of device information and configured/edited information to XML files for runtime use on both x86 and x64 Windows.

ACKNOWLEDGEMENTS

The authors appreciate useful advices from T. Scarvie, and the system management help from C. Ikami and T. Kellogg.

REFERENCES

[1] L. R. Dalesio, et al., ICALEPCS '93, Berlin, Germany, 1993. http://www.aps.anl.gov/epics/

[2] LBL PUB-5172 Rev. LBL, 1986.

A. Jackson, IEEE PAC93, 93CH3279-7(1993)1432

[3] http://www-controls.als.lbl.gov/epics_collaboration/sca/

[4] C. Timossi and H. Nishimura, IEEE PAC'97, 0-7803-4376-X/98, p805, 1998

http://www-controls.als.lbl.gov/epics_collaboration/sca/win32

[5] H. Nishimura and C. Timossi, PCaPAC 2005, Hayama, Japan, 2005.

[6] H. Nishimura and T. Scarvie, EPAC 2006, Edinburgh, Scotland, 2006.

[7] Jeff Hill, http://www.aps.anl.gov/epics/contacts.php

[8] http://www.amd.com/us-en/Processors/TechnicalResources/0,,30_182_871_13118,00.html.

[9] http://zedgraph.org

[10] D. Icardi, http://www.devage.com/

[11] http://www.mysql.com/

A TUTORIAL ON PROJECT MANAGEMENT

J. Kamenik, P. Kolaric, M. Plesko, I. Verstovsek, Cosylab, Slovenia.

Abstract

Just be clear about one thing first: project management is not entering random dates and numbers into Microsoft Project and printing them on long rolls of paper just to make management happy. There is more to project management than meets the eye...

As all disciplines, project management has a set of rules that must be followed and set of recommendations that make work easier. But as in all engineering, there is no single magical formula or equation, no matter how much managers and physicists alike would love to have it. Yet, fortunately, it is in a way a "linear theory", as it can be broken down into small steps that can be followed by a convenient check list.

Steps and check lists will cover the first part of our tutorial, where their meaning and use will be explained on a real example such as an accelerator control system. Although all items are strictly common sense, there are so many of them that one cannot be aware of them all of the time - hence the other common sense trick of using check lists. Don't worry, although our simple check lists contain nearly a hundred items, we will concentrate on those that are important and simple but often overlooked, especially by those that like to do projects in the infamous zero-th order approximation: "code first - ask later".

The second part of the tutorial will discuss 2nd order effects, which all honest and respectable physicists, like myself, prefer to ignore anyway. Among them are risk analysis and risk management, which sound like another group of those fancy buzzwords, but in reality mean just this: "think about what could go wrong and what you are going to do about it before it happens, and then do it immediately when it happens". Other popular, or better unpopular, 2nd order project management effects that we will discuss are resource planning, handling interaction among team members and with the "client", change and version management, analyzing data on project with quantifiable metrics (e.g. work spent vs. estimated), systematic search for common errors and possible sources of confusion (yes, check lists again), using a database of finished projects to help manage current projects, and more.

INTRODUCTION

When a serious scientist (or most of other institute members) is asked about the purpose and usefulness of project management, the thoughts usually go along the lines of *"Entering random numbers into Microsoft project and printing long rolls of paper", "Filling pointless reports nobody looks at", "Something that makes management happy", etc.*

However, the same people are very hard working in nature and therefore tend to get overbooked with work, are not sure on what to focus because "everything is urgent". The purpose of project management tools and techniques is to prevent these kinds of situations.

Not all project management techniques are of the same relevance to research institutes. In this paper we will focus on the most critical ones.

PROJECT PHASES

In a research environment the most crucial step of a project is not the actual implementation - scientist are very good a doing things: thinking, coding, debugging, etc. and usually they perceive this as fun. The critical parts of projects are the things around the fun stuff, in particular:

- conception (evaluating the idea before jumping into implementation)
- planning (how to get things done in terms of time, resources)
- closing (to finish all work on a project without a need for constant support)

From Idea to Proposal

The main pitfalls here are that:

- projects are started too easily (without basic thinking about purpose and scope),
- too many projects in parallel: focus of work is lost.

What one can do about it is:

- **a quick sanity check**: why start something? Think about the big picture as well. How will this help the work on other projects of the institute, etc.
- Prepare a **well rounded proposal**. By putting things in writing motivates you to think also about the not-so-fun stuff.
- Use **formal decision process** to start a project. This does not have to be a meeting with all the managers of an institute. Presenting the proposal in a well prepared presentation to your group colleagues can do the trick as well. During the discussion a decision whether the project is worth a shot or not will most probably spring up by itself.

See [1] for some checklists that apply to this stage in a project.

Planning

The pitfalls of not doing planning are:

- people get over allocated and ineffective,
- there is no reference to track progress,
- there is no “satisfaction of a job well done”.

What to do about it:

- prepare **at least the initial project plan.** Even if the plan is never updated later, there is at least one reference point to relate to.
- Make a **risk plan**.
- **Communicate the plan** and get commitment. This very important and often neglected: the plan will hard to carry out if your team does not about or if the plan is not accepted.

A checklist for identifying risk is so important that we state it here:

- A short description of the risk.
- When it is expected to occur.
- The probability assessed.
- What consequences are expected.
- What actions you will take if it happens.
- Who will take the actions.
- Who is responsible for monitoring the risk.

From the checklist make a risk log and update it regularly, at least once a month. For other checklists for planning, see [1].

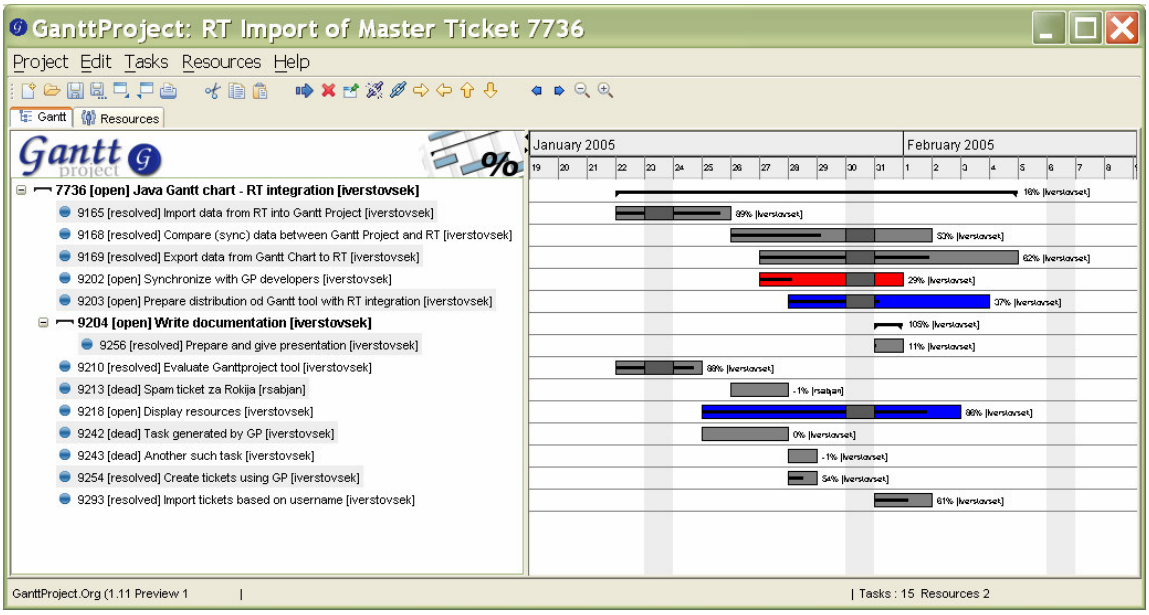


Figure 1: Gantt Project with tickets, imported from the Request Tracker.

Closure and Evaluation

This is the "sugar on top" that should come after the work on a project is coming to an end. Closing the project formally prevents the never-ending work and allows the whole team to learn from past experience on the project. You should at least take care of two things:

- **Acceptance process.** Acceptance should be made for internal projects as well - these projects are the most prone to an eternal life.
- Close project, **plan resources for support**. Important here is that the resources for support are well defined and are planned for a limited period of time.

Again, see [1] for the relevant checklists.

TRACKING PROGRESS

The Cultural Issue

In a scientific environment, one is expected to do research - to work on projects that do not have a well defined flow and consequentially, one cannot estimate how much work will be spent. Along these lines it does not seem make sense to enter time report time work.

However, entering time and reporting progress of work can be helpful - it can assist the overall planning of activity and therefore reducing the future stress of the scientist. The scientist will use the system only if he or she sees a clear benefit in doing so. For any reporting and management system the following must be true:

- rhe system should introduce minimal overhead to its user,
- it should display its benefits fast.

Define Basic Units of Work

Any project is composed of a number of more-or-less well defined units of work, assigned to developers, usually by the project managers. There is a multitude of tools that manage "work units", many of them freely downloadable from the Internet. We chose the Request Tracker (RT) [2] because of one simple reason – this tool can manage e-mails really well – not only sending but also receiving. RT was picked as our main tool and we adapted all the other ones to it. Every now and then, when experiencing problems with RT's code written in Perl, a question appears why we did not rather write such tool by ourselves, but the final statement remains that RT is really well structured and extremely useful.

RT is (as described in its manual) an "enterprise-grade ticketing system which enables a group of people to intelligently and efficiently manage tasks, issues, and requests submitted by a community of users".

Its main unit is a ticket, which represents a specific task to be done. It has several fields:

- status (new, open, resolved, stalled, dead),
- estimated time,
- time spent (increases whenever a user reports a transaction),
- dates: start date, due date,
- priority,
- keywords (severity of a bug, type of ticket: QA ticket, Master ticket for the project, etc.), and
- links - how does this ticket relate to other tickets (parent - child, "depends on" and "refers to" relationships are supported).

Each ticket can have one or more parents, children or brothers. Setting also dependencies, a clear structure can be made. RT can warn us by email of a creation or modification of some ticket. The best feature of RT is the possibility of managing tickets via e-mail – every project has its e-mail address to which we can send a request for creation, correspondence or comment and set just about every field of the ticket.

RT has easy-to-use search functionality, which allows one to quickly find a ticket. With all its features RT can be used for handling support requests, ordinary tasks and bug tracking. We create about 10.000 tickets per year in Cosylab.

In Cosylab, the system is used to track time in minutes. This means that after finishing working on a given ticket, the developer must write a reply to RT on what he or she did and how long did it take. In order to reduce overhead for this activity, we have implemented a stopwatch application: it is an icon in the system tray, when clicked upon, it displays a list of all your tickets from which you select one. When switching a task, you switch it also in the stopwatch. At the end of the day, you just need to commit all the recorded times to the db and write comments where necessary.

In DESY, the goal of the system is not to track the time spent of individual developers, but to track their *progress* on the tasks. It was not necessary to modify the system a lot to accommodate for this functionality - instead of entering time, developers enter their progress in relative units, where 100% is a completed task.

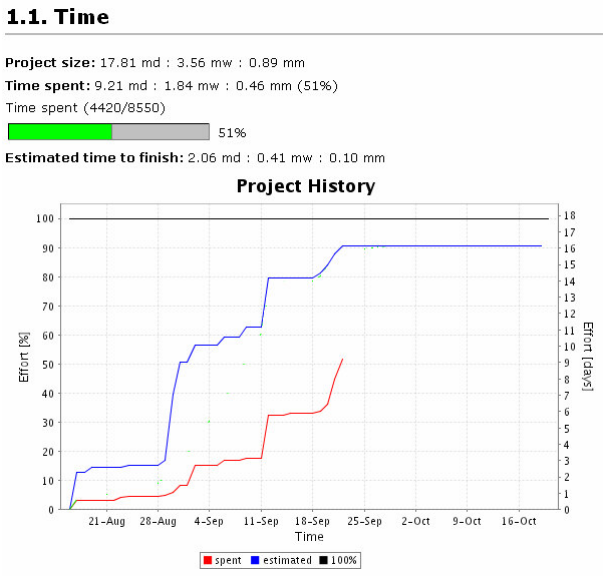


Figure 2: Work spent on a project versus time. Blue line: planned use of work, red: actual time worked.

CONCLUSION

In this paper we have presented a few of the key issues of project management when applied to research institutes. Due to the lack of space we have omitted very important concepts such as resource planning, handling of team dynamics, communication with the client, etc.

This paper will more than fulfil its goal if it has persuaded you to do at least some of the following:

- take extra time for conception, planning and closing stages of projects,
- manage risks thought the project,
- use some sort of project management / reporting system,
- monitor project progress (close the loop),
- ease the "cultural issues" by demonstrating benefits of project management to your team and by constant coaching.

REFERENCES

[1] Successful Project Management, Trevor L. Young, Konan Page Limited, 2000

[2] I. Verstovšek et al., Cosylab Management Sytem, PCaPAC 2005, Hayama, Japan.

STANDARDISATION OF THE PSI ACCELERATOR CONTROL SYSTEMS

D.Anicic, T.Korhonen, A.C.Mezger, D.Vermeulen,
Paul Scherrer Institut, Villigen, Switzerland.

Abstract

At the Paul Scherrer Institute (PSI) several accelerator facilities are run from the central control room. Two control systems are used for the machine operations: the SLS (Swiss Light Source) based on EPICS and the high intensity proton accelerators facility based on ACS, an in house developed Accelerator Control System. PROSCAN, the biomedical proton facility uses the latter system (ACS), too.

The decision to standardize the hardware equipment as much as possible has already been taken a few years ago and is widely taking place. However the effort to maintain and continuously develop two control systems is unnecessarily consuming precious human resources. Therefore the proposal has been made to implement EPICS at the proton facilities besides ACS. The two systems will run concurrently until hardware transition towards VME has been completed, building this way a migration path towards EPICS, enhanced with many ACS features.

INTRODUCTION

Currently three of PSI accelerator installations use two control systems. The SLS is completely EPICS based, with about 200 VME crates. With the installation of remaining beam-lines this number will increase by 100. The high intensity proton accelerator complex, with three proton cyclotrons, is based on in-house developed control system, called ACS. Control and diagnostics systems hardware is mainly CAMAC with approximately 100 crates. CAMAC crates are accessed through 15 VME based front-end computers (FEC, further on referenced as IOC). The upgrade/modernization started already 3 years ago, with five VME crates for 59 power supplies and 13 digital beam position monitors. The third accelerator system, for mainly medical cancer therapy purposes, is also ACS based, with only VME based hardware. In the final state (End of 2007) it will consist of about 20 VME crates.

MOTIVATION

Although EPICS could easily replace ACS without hardware changes, the main obstacle remains the aging CAMAC hardware components. Some are already 30 years old. Despite it has not been observed already, it is expected that CAMAC equipment failures will increase in the future. Obtaining spares or spare parts to repair them is an ever greater challenge. Due to retirements, expertise gets lost too, and it's much easier to find VME developers and parts and support from commercial companies. Replacing CAMAC with VME will thus reduce failures and save resources in a longer term. In addition to it, the beneficial man-power savings could be achieved by standardizing the control system software, too. Migration of ACS control systems of both proton accelerator facilities towards EPICS is about to commence.

ACS FEATURES

ACS basic operating principles are probably very similar as in many other control systems. It is distributed and client-server based. Client side applications, like operator displays or even closed loop regulations, communicate requests over the local area network to the server computers, the IOCs. Starting 17 years ago with PDP-11 servers and VAX workstations (backends) over raw Ethernet, towards today's HP-RT and LynxOS real-time VME single board servers (IOC) and Linux workstations over TCP/IP, are used. Only CAMAC devices were initially supported, but since approximately 3 years VME and PLC equipment are implemented too. All these upgrades have been done in small steps over the above mentioned period, with manpower of 3 to 5 persons.

From the beginning, all of the IOC configuration data and backend utility data (CoreDB, for name discovery and others) have been tightly coupled to the control system core. Initially using files and proprietary data-administration applications, and shortly afterwards was switched to an Oracle database and its tools. It is practically impossible to maintain ACS operational without it. Simply said, all configuration data is stored and retrieved from database.

ACS assumes a strict naming convention of Device, Attribute and Conversion. Due to historical reasons those are limited to eight characters for a Device and four for an Attribute. A Control system object consists of one or more attributes, representing together all the data for a particular "device". There are up to three conversions, 1 for DAC values (Unit), 2 for engineering units (V, A, mm, Hz, …) and 3 for physical units.(kGauss, mBar, …). There is a variety of attributes, representing, for example, status (STA, STAX), commands (COM, COMX), read-back value (IST), set value (SOL), high and low limits (HL, LL, …), and many others.

CoreDB data and associated libraries provide name discovery for applications. Search methods are available for almost all properties using wild-cards. So it is possible to obtain a list of all Devices and Attributes, by choosing a particular Device or Attribute pattern, system it belongs to (diagnostics, vacuum, run-permit-system, beam-optics, …), area in the beam lines, building/room, type (dipole, quad, bpm, profile-monitor, beam-stopper, …), software driver it is handled by, and others. That makes it possible to write dynamically- configurable user/operator applications.

ACS libraries and CoreDB additionally offer methods for second level data evaluation. Here we basically consider processing for which we believed does not belong to the IOC, but rather at the backend level. Status-bits interpretation logic, resulting in a clear text human usable result is one of these. All the necessary data for it comes from Oracle database.

ACS offers RPC gateways for MS-Windows users. The device IO with access control, and archived data retrieval are provided for LABVIEW and Visual Basic applications.

A modified MEDM version using the latest MEDM version (3.04.B6), supporting our CoreDB features (like automatic selection of animated gif images based on second level status-bits interpretation) is available. The MEDM add-ons are implemented through the medmCdev.c connectivity interface.

An extra feature of ACS, mostly not found in other control systems, and implemented at IOC level, is support for atomic-increment (and some other) actions, besides usual read (get) and write (set) actions.

ACS ADDONS FOR EPICS

EPICS is a widely used control system toolkit. As it has already been used successfully at our SLS facility, it makes a very good starting point for being used at the proton accelerator facilities, too. By being actually a toolkit for building control systems, it has to be adapted to the given circumstances.

The most important is the naming convention, that has to suit the ACS style. Operations and other users should not be negatively affected by having to learn new names. Everything has to continue to work the same way or at least in a similar way. Special consideration has to be given to the presence of conversion levels. The most probable solution is to have names tripled, in a form

DEVICE:ATTRIBUTE:CONVERSION

(like QXA1:SOL:1, QXA1:SOL:2, QXA1:SOL:3). Another very important reason to keep the same names is that they are widely used for labeling cables and devices.

Although there are already CAMAC drivers for EPICS available, the one we use (from CERN) is not supported. The driver would have to be implemented, but this effort will probably be skipped.

Several VME boards already used at ACS, PROSCAN and SLS are the same, but in most cases with specific application required firmware version. Those EPICS drivers will have to be written, too.

In ACS we rely mostly on just-in-time access to hardware registers, in contrary to EPICS which is based on buffered IO (either scan or interrupt). The estimate is that such a behavior will be implemented in EPICS records through the PROC attribute.

Special ACS actions (atomic-increment) will have to be implemented in EPICS records. This means additional records to the tripled as mentioned above. To provide unified naming, the above naming convention would probably have to be extended by adding a postfix ‘INC’ to the conversion level.

In view of the fact that no applications are implemented at IOC level, we hope to find a way of generating EPICS records directly form our existing Oracle database.

The existing CoreDB (name discovery, status-bits interpretation and others) have to be on disposal for EPICS applications, too.

The Existing RPC gateways will be adapted to EPICS Channel-Access for user convenience.

EPICS INTEGRATION PATH

The migration to EPICS still has to be carefully planned into details. The PROSCAN facility is the first to be migrated. It is less complex, and already has no CAMAC equipment, being fully VME and PLC based. However it will not be easy to find the right moment to do this, the facility being full time operated with patients from the beginning of December of this year. The common part will be worked-out; the later ACS migration would just require additional drivers. ACS migration towards VME has already begun, but would need considerable time. The migration process has to be scheduled in regular shut-downs, without affecting beam production.

Phase 1

Replacing CAMAC equipment with VME will be done smoothly. It will probably take 5 years.

Mixed, ACS + EPICS, control system will be used during that time.

In order to support the dual control system, a two-way gateway has to be implemented.

ACS features (CoreDB, …) for EPICS applications is to be implemented.

EPICS records for additional ACS features must be supported.

EPICS records have to be generated from existing Oracle database. This also implies database modification to differentiate between ACS and EPICS devices.

Phase 2

Modify high level applications to use EPICS Channel-Access, plus ACS features (CoreDB).

The two-way gateway will phase out only when all CAMAC hardware has been replaced with VME and all applications have been ported.

The ACS features (EPICS add-ons) will be part of PSI EPICS.

CONCLUSIONS

A lot of work is in front of us. A lot of money will be spent on CAMAC replacement (which has to be done anyhow). The benefits will be visible only in years to come after. Still, we are confident that the merits will be multiple, not only for Controls, but for all supporting groups and users, too.

STATUS OF ISAC CONTROL SYSTEM

Chris Payne, ISAC - TRIUMF, Vancouver, BC, Canada

Abstract

The ISAC facility at TRIUMF has been delivering radioactive isotopes to experiments since late 1998 using an EPICS based controls system. Initially at ISAC this was a combination of Solaris servers, VxWorks on Motorola 68000 based IOCs, and Windows 98 Operator interface stations. The ISAC controls network connectivity was originally simply the TRIUMF public network.

The ISAC facility has expanded considerably in the following years and now includes complex ion sources, both room temperature and superconducting accelerators, as well as numerous, complex experimental facilities. As the ISAC facility has expanded in both size and complexity, and the size of the user community has increased, the limitations in the initial controls configuration have become apparent. In order to cost effectively deal with these issues, many changes have been made in the hardware and software in use at the ISAC facility.

This paper will discuss the migration of some primary ISAC controls services to less expensive Intel/Linux computers as well as the isolation and segregation of the ISAC controls network to ensure a robust and secure ISAC Control System.

HISTORY

Control System

The TRI University Meson Facility (TRIUMF) Central Control System (CCS) was initially classic dials and knobs hard wired back to a central control room. The CCS evolved over time as computers became more readily available, eventually migrating to VMS running on MicroVAX's, then DEC Alpha's and finally Intel Itanium's. The CCS has proven its stability and reliability through many years of use at TRIUMF.

Figure 1: Cyclotron Control Console circa 1974

In 1995 the Isotope Separator ACcelerator (ISAC) project at TRIUMF was funded, and shortly thereafter the decision made to implement the control system using the Experimental Physics and Industrial Control System (EPICS). EPICS was initially developed by a few US Department of Energy accelerator labs, but has since evolved into a large worldwide collaboration, with many sites participating actively in the development. [1]

EPICS is a modern, distributed control system which has been designed with network control in mind from initial conception. EPICS is a mature control system which can be used to effectively control geographically separated systems, requiring only network links for communication between the Operations Staff and the devices they control.

Network

TRIUMF predates widespread computer networking and as such the network infrastructure at TRIUMF was an after thought, being deployed well after the initial CCS implementation. Inevitably networked controls equipment was introduced to TRIUMF and over the first few years the network expanded in an arbitrary and somewhat haphazard fashion.

The ISAC controls network was first implemented on the TRIUMF network as part of the common, flat address space. Although initially convenient for developers, the reality of reliable operations necessitated some kind of segregation of the network layout which is described below.

LINUX ON PERSONAL COMPUTERS IN ISAC

Linux Workstations

The ISAC Operator workstations in 1999 were Intel PC's running Windows 98, with SSH and X Window clients used for access to controls. This setup was chosen due to the ease with which multiple monitor desktop setups could be created. The EPICS development and production servers in ISAC in use at the time were Solaris based Sun workstations.

This setup, although acceptable during commissioning, did not prove reliable (stable) enough to be a viable 24x7 operations configuration. The Windows PCs were chronically unreliable, and reboots occurred several times per day. With Linux rapidly becoming mature around this time (2000), it seemed a natural choice as a replacement operating system for the ISAC Operations consoles.

There was some initial user (Operator) resistance to the new operating system as users are generally familiar with a Microsoft environment from personal use, and anything which varies slightly from this is met with resistance. However, after only a few weeks of stable system

operation, most were in agreement that the move to Linux was a logical and natural progression, and all ISAC Operations consoles were migrated to Linux.

Initially, the Linux consoles were merely serving as X Windows displays for the applications running on the Solaris servers. Migrating EPICS client applications from the Solaris server to the Linux workstations was an obvious move.

The Linux version in use at ISAC has continued to be updated, and most recently has been standardized on Scientific Linux (SL), a release created out of a collaboration between Fermilab and CERN. [2] TRIUMF maintains a local mirror of Fermi SL which allows for easy initial installation as well as subsequent updates and patching.

Figure 2: ISAC Operator Console circa 2006

Linux Firewalls

The operation of the ISAC facility at TRIUMF, like all recent accelerator facilities, is "fly by wire", where there are no direct connections between the Operator interfaces and the devices they are controlling. Network connections therefore become mission critical, so that both the network hardware and the associated wiring must fall under tighter control than the public TRIUMF network.

The first step in securing the ISAC controls network was to reconfigure network in order to amalgamate and isolate all required devices on a physically separated network. This move made possible a single connection between ISAC controls and the TRIUMF site network and by extension the Internet. All that was required was a means to filter the traffic through this connection, and a Linux bridging firewall was the natural solution.

The configuration of the TRIUMF network at time of ISAC firewall installation was a single flat class B address space, with all computers existing in a single broadcast space. As a result of this shared address space, network disturbances such as virus or worm outbreaks, as well as random broadcast storms could negatively affect ISAC controls performance, a situation which was unacceptable. By using a Linux Ethernet bridge with net-filtering enabled [3] the IP traffic to the ISAC controls network could be filtered in an intelligent manner without disturbing the legacy TRIUMF network layout. Adding an additional package called Ebtables [4] allowed all extraneous Ethernet packets to be blocked at the perimeter of the ISAC controls network.

Although initially exclusively non-Windows, the scope of the ISAC Control system has grown over the years and many diverse types of controls equipment have become a reality in ISAC. The first major installation of non-EPICS controls was the Radio Frequency (RF) controls running on Microsoft Windows PCs. In order to protect the RF controls from the Internet, while still protecting the ISAC controls network proper from the perceived problem computers, a third interface was added to the ISAC controls firewall. The new interface allowed for the creation of a classic De-Militarized Zone (DMZ) where non-standard controls computers could exist in a semi-protected segment, isolated from the main ISAC controls network. Unfortunately, in this configuration the ISAC controls firewall becomes a mission critical system. A standby "warm spare" firewall computer is maintained to facilitate minimal downtime in the event of hardware failure.

Finally, in early 2006, the TRIUMF network was reorganized such that the ISAC controls network was officially separated onto its own Virtual Local Area Network (VLAN). This separate VLAN had some unforeseen side effects, but has increased the overall integrity of the ISAC controls network significantly. To facilitate communication and monitoring of the network infrastructure equipment, the ISAC controls firewall had to be modified to make it VLAN aware.

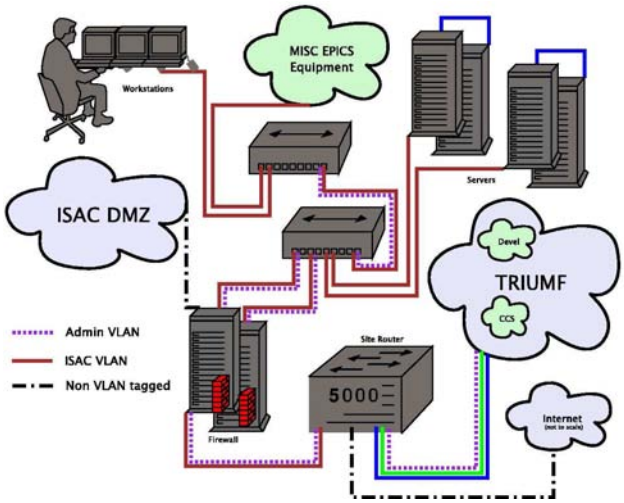


Figure 3: Network Functional Schematic

The choice of PC hardware with Linux OS for the ISAC firewall device was seen as somewhat controversial. Throughput tests and latency monitoring before and after all transitions of the network and firewall layout have determined that PC hardware which had become obsolete as far as desktop use was easily powerful enough for even complex filtering of high volume network traffic. The benefits of a firewalled ISAC controls network, at the price of merely time invested, are therefore easily justifiable.

Linux Application Servers

The ISAC EPICS user community continued to increase in size as the facility continued to grow, and the limitations of the Solaris application servers were starting to be reached. The required EPICS client applications had already been compiled for Linux as a result of the prior work for the ISAC Operations consoles, and therefore the migration of general EPICS users from Solaris to more powerful, yet less expensive PC hardware running Linux proved trivial.

A second concern raised by the growing ISAC EPICS user community was the increased load on the Input Output Computers (IOCs) in the field. To transfer the additional computing and network load off of the IOCs, the EPICS Process Variable Gateway [5] was installed on the Linux application server at the time of migration. Beyond transferring CPU and network load, the EPICS Gateway also allows much easier configuration of access control of the ISAC EPICS users.

Redundancy

The original Solaris servers were redundant, and in the event of hardware failure could in theory replace each other. However, the migration in emergency situations was a non-trivial task involving service start on the replacement machine and offline work to synchronize data files. With the cost of the PC hardware being much less than the "cost" of experiment downtime, an identical PC was purchased at the same time as the Linux Application server installation to be used as a fully redundant spare. In order to completely minimize switchover time, the redundant computer is configured almost identically, right down to the IP address. Synchronization between these identical machines is performed hourly over a private network connection. In the event of hardware failure, the only required intervention is to take the primary computer off the network and move the secondary computer to the public network. Users may then merely login again as if only a minor network glitch had occurred, and the failed system can be investigated offline and without undue pressure to restore services. This system is termed "warm spare" locally and has been proven to work as described above, with the ISAC Controls group successfully performing test switches in a controlled fashion, as well as switches during a real hardware failure.

Linux Web Servers

The Apache web server has been the most popular web server on the internet for more than 10 years [6], and its stability is well proven. With this in mind, it was an obvious choice for ISAC Controls and ISAC Operations to migrate their web services from Windows NT to Linux Web Servers for all mission critical systems.

Access to some of the ISAC web services is authenticated via a TRIUMF email username/password pair. This mechanism provides a convenient way to allow all registered TRIUMF staff authenticated access. Running a slave Lightweight Directory Access Protocol (LDAP) server locally on the ISAC web server minimizes availability downtime without losing the convenience of the common username/password.

As with the Linux application servers, the ISAC Linux web servers are run with "warm spare" counterparts in order to minimize downtime in the event of hardware failure.

Linux File Servers

The bulk of the file serving required prior to 2000 was for developer use from Microsoft Windows PCs. The initial natural fit for this a Windows server, specifically a Windows NT file server.

However, after the migration of other services to Linux, the natural progression was to a Linux file server using Samba [7] to allow Windows PC access. Once initially configured, this has proven a reliable and low maintenance solution.

Other basic file server requirements of the controls system for Operator documentation and data sharing are accomplished using NFS, a well established protocol. Both Solaris and Unix servers serve files to clients.

Linux Database Servers

Prior to 2000, the database used by ISAC Controls was Paradox, and similar to the Web Servers and File Servers it was running on a Windows server. As other services such as web serving and file serving migrated to Linux, the database server was moved to the new platform as well. After comparing viable Open Source alternatives, PostgreSQL [8l] was chosen as a replacement database management system. PostgreSQL is mature, stable, and fully featured and has proven to be an excellent database management system for ISAC.

SUMMARY

The use of personal computer hardware along side freely available open source operating systems and applications has proven to be an intelligent choice for ISAC. While the costs have been kept to a minimum, the overall system stability and availability has been increased enormously.

REFERENCES

[1] EPICS http://www.aps.anl.gov/epics/

[2] Scientific Linux https://www.scientificlinux.org/

[3] Linux Bridge netfilter http://linux-net.osdl.org/index.php/Bridge

[4] Ebtables Ethernet filtering http://ebtables.sourceforge.net/

[5] Gateway: The Process Variable Gateway http://www.aps.anl.gov/epics/extensions/gateway/index.php

[6] Netcraft Web Server Survey http://news.netcraft.com/archives/web_server_survey.html

[7] Samba homepage https://www.samba.org

[8] PostgreSQL homepage http://www.postgresql.org

FIRST OPERATION WITH SPARC CONTROL SYSTEM

M. Bellaveglia, G. Di Pirro, D. Filippetto, E. Pace, INFN-LNF, 00044 Frascati (Italy)
L. Catani, A. Cianchi, INFN-RM2, 00133 Rome (Italy).

Abstract

The SPARC[1] gun and the diagnostic apparatus called emittance meter (e-meter) have been installed in all components. The complete installation of SPARC accelerator is planned for the end of 2006.

The first part of the installation allows to test the architecture of the control system from the hardware and from the software point of view. Control application for magnetic elements, vacuum equipments, RF cavities and some diagnostics have been developed and debugged on line. In order to improve the machine operations we have included in the system some operation service.

An electronic logbook has been used since the first phase of the operation contributing to share the information between all the members of the collaboration.

We began to develop an automatic system the accelerator status periodically or when some value changes. This system is based on a PostgreSQL database server.

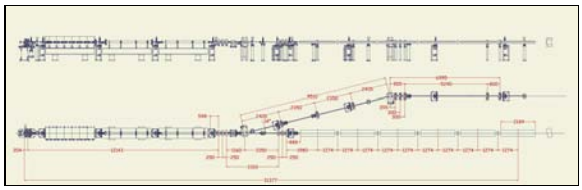


Figure 1: SPARC Layout

SPARC

The SPARC (Self-Amplified Pulsed Coherent Radiation Source) (fig.1) project is to promote an R&D activity oriented to the development of a high brightness photo injector to drive SASE-FEL experiments at 500 nm and higher harmonics generation. Proposed by the research institutions ENEA, INFN, CNR with collaboration of Universita` di Roma Tor Vergata and INFM-ST, it has been funded in 2003 by the Italian Government with a 3 year time schedule. The machine is under installation at Laboratori Nazionali di Frascati (LNF-INFN). It is composed of an RF gun driven by a Ti:Sa laser to produce 10-ps flat top pulses on the photocathode, injecting into three SLAC accelerating.

The e-meter

The gun has been installed with a diagnostic apparatus called e-meter [4] (fig. 2). This apparatus allows us to characterize the first 2m of the electron beam. The main component, from the control system point of view, is the emettance measure apparatus composed by a pepper-pot and a YAG target. This part of the e-meter can be moved in any position along a 2 m bellow. At the end is available a spectrometer to measure the energy and a toroid to read the bunch charge. In tab.1 the different component are shown.

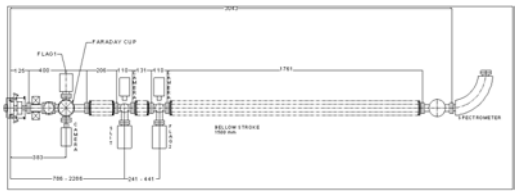


Figure 2: E-meter Layout

SYSTEM CHOISE

The SPARC Control System is in charge of managing devices (Tab. 1) distributed over an accelerator area. To develop the whole system we have short time and few people. We decided to use commercial technologies as much as possible in order to optimize the development time. A commercial product is characterized by a broad distribution, which means a lot of feedback from the users and, consequently, deep debugging. Furthermore the wider is the distribution of a product the more reliable is its support from the producer

Another criterion was to privilege "easy development and maintenance".

We decided to use:

- LabVIEW from National Instrument is used development as environment for all the software;
- Industrial Personal Computer with PCI bus to house the front-end hardware

SYSTEM DESCRIPTION

The main operation in an accelerator control system is data taking, display of information, analysis, command execution and expandability. To reach these goals we need to use a well defined system structure. We chose a simple but efficient three levels architecture.

Console level it is the human interface. Several equipollent consoles, built on small personal computers with Linux as operating system.

Service level is the second and central level of the system. It essentially contains a CPU that acts as a general concentrator and coordinator of messages throughout the system. We log automatically the command, the machine status and the errors. A second processor is used at this level with an SQL database to store automatically the information from the front end processors.

Front-end level is constituted by some (more than 8) industrial Personal Computer distributed round the machine. Each PC performs control and readout of an element of the accelerator. The information can be read by the console on request.

Device	e-meter	SPARC	Interface
Magnet P.S	9	30	serial
Vacuum pump	15	23	Fieldpoint
Vacumeter	2	6	serial
Modulator	2	2	Ethernet
RF	2	2	LAN, Digitizer
Camera	5	12	IEEE1394
Flag	5	12	Serial, CAN
Current Monitor	1	2	Multimeter
Position Monitor	0	12	Ethernet

Table 1: Elements

In a distributed system the interconnection bus between the different CPUs is important to allow maximum performance. First of all we don't want to have any evident bottleneck in the data transfer. Furthermore the bus system has to be reliable and affordable. Today in every personal computer the Ethernet connection is a standard: this means that it can be a robust channel of communication. We use the Gigabit Ethernet to obtain the necessary bandwidth in the data transfer between the different parts of the system.

The realization of a switched LAN gives the possibility to use the network also as a fieldbus infrastructure to reduce at maximum the interconnection between the devices and the acquisition system. In table 1 we can see the elements of the acquisition system: some of them can be directly connected to Ethernet some other fieldbus can easily connected to it.

SOFTWARE

In order to reduce the time of development of the SPARC control system, we decided to use well known software. Labview became the natural choice for the following reasons:

- in the Frascati laboratory the use of National Instrument software is diffused (we can say it is a "standard");
- Labview is used as development software in the DAFNE control system [3]. This choice allows us to re-use, when possible, the software;
- Labview is considered reference software by a lot of hardware manufacturers that write interface divers in Labview.

The DAFNE control system is working since 10 years and now is well defined and debugged. This encourages us in using the same architecture and software, when possible, for the SPARC control system.

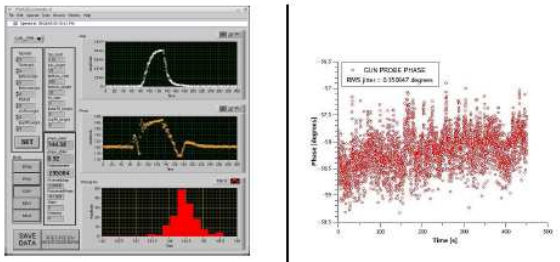
Analyzing the two systems we have found two main differences: first of all the acquisition bus is PCI for SPARC VME for DAFNE. Furthermore the communication between the system levels is different. For these reasons it is necessary to rewrite some software.

We began the porting of the software starting from the communication mechanisms. In DAFNE the shipment of the commands happens using a mailbox written on a shared memory. In the SPARC control we do not have a shared memory but a LAN. A server program, running in parallel, receives the commands and makes them available to the acquisition program through a global variable. In the DAFNE control system a second communication channel is present that allows reading the state and the variations of the single elements under control. Also this communication is through shared memory. In the SPARC case we have realized a second server that bundles the information and sends them upon request to the console. Presently the data are transferred without coding them but we are thinking of using XML coding system [5] in the future.

The usage of PCI bus instead of VME, in the front-end CPU, forced to replace the acquisition board drivers. In some cases this was not necessary because the acquisition happens through a secondary fieldbus (Tab.1). This allowed simpler re-use of the DAFNE software to that element.

Radiofrequency

A new acquisition system for radiofrequency (RF) signal monitoring and synchronization is designed as a fundamental part of the SPARC project at LNF and it is currently working with very good performances.



(a) (b)

Figure 3 Control system application to measure the phase noise

The core of the synchronization system is a demodulation board and digitizer cards in an industrial PC, where data analysis and device control are accomplished. The above apparatus can be seen as a custom multi-channel digital scope, able to display in the control room all the demodulated signals coming from the RF structures placed along the whole machine. The waveform is digitized using data acquisition (DAQ) cards that are 12bit 60Msamples/s A/D converters. This system allows a real time monitor of amplitude and phase of the RF pulses along the machine.

To accomplish the phase noise monitor task, we avoid transmitting the whole acquired waveforms from the tunnel to the control room. To implement a shot to shot monitor at the 10Hz repetition rate of the machine, we analyze data inside the same software application running

in the front-end industrial PC. In that way, only a number, representing the phase for each location of interest, can be sent to the control room. Moreover, the control system has been designed to perform a "one-click" phase noise measurement along the whole machine. A snapshot of the console application is shown in figure 3(a) and results relative to the RF gun phase noise are shown in figure 3(b).

To reduce the phase noise and to enhance the performances of the photo-injector, we implemented also a phase feedback that analyze the acquired values and controls a motorized phase shifter to compensate slow drifts figure 4.

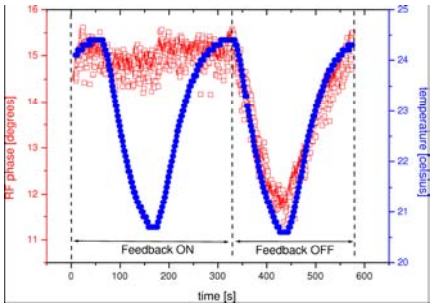


Figure 4 Compensation of the slow phase drifts

Diagnostic

The main machine parameter emittance, bunch length and energy in SPARC are measured with images. The use of a versatile camera system is strategic in the realization of this diagnostic. The rapid evolution in the image acquisition systems allows us to choose the camera and its own interface in a wide variety of products. The IEEE1394 interface gives us the possibility to interface different camera with different specifications without changing the software

The cameras are acquired by different distributed personal computers that send data trough a TCP/IP channel. We well defined the data transfer structure to full integrate the cameras inside the control system.

Another important component in the diagnostic is the control of motors to move flags and slits to allow the acquisition of the beam image. Also for the e-meter we need to move position slits and flags.

We have written some useful programs to acquire automatically the position and the dimension of the image of the beam and to save them. The saved images are used by offline beam analysis.

SERVICE PROGRAMS

The SPARC collaboration involves different research national and international research institutions. Some services are necessary to allow all people to have the information available on the status of the machine and the progress of the work. The old system based on a logbook where the operator writes the data and glues picture on it can be useful but cannot be available from remote researchers. We choose a freeware electronic logbook that we have customized. This choice is in easy to install and to use ELOG seems the good choice.

Status log machine

The injector project is an experimental machine the possibility to have an automatic saving mechanism can be useful in offline analysis.

We are studying and developing a data acquisition system based on a database with a possibility to communicate via TCP/IP. We decide to use the PostgreSQL database.

For the moment on each front-end processor a database client program periodically saves the data of the controlled elements. Some interface to plot historical data at console level have been developed.

The system is currently acquiring information by all the elements of the e-meter apparatus. Performance of the system is under test.

STATUS OF THE ART

During the test of the e-meter the control system software has been completely defined, implemented and tested. We also implemented and started the test of the machine status log.

We plan some future developments of our control system with the introduction of some embedded systems to monitor some other elements. We started the study how to guarantee the synchronization of different PC in the system.

ACKNOWLEDGEMETS

We want to thank F.Anelli, S.Fioravanti, L. A. Rossi and S. Strabioli for their contribution in the software development and hardware installation.

We are also grateful to all the SPARC staff for their continuous suggestions and encouragement for making a good and useful job.

REFERENCES

[1] The SPARC project is financially supported by the EU Commission in the 6th FP, contract no. 011935 EUROFEL and contract no. RII3-CT-2003-506395 CARE.

[2] SPARC Project Team, Sparc Injector TDR http://www.lnf.infn.it/acceleratori/sparc/

[3] G. Di Pirro et al. "DANTE: Control System for DAFNE based on Macintosh and LabView", Nuclear Instrument and Methods in Physics Research A 352 (1994) 455-475.

[4] A. Cianchi, et alt : "Design Study of a Movable Emittance Meter Device for the SPARC Photoinjector", Proceedings of EPAC2004, 5-7 July, 2004 Lucerne, Switzerland;

[5] L. Catani, "A Communication Protocol for a Distributed Control System with LabVIEW", PcaPAC2006, October 24-27,2006 CEBAF Center Jefferson Lab Newport News, VA USA

EPICS ARCHIVEVIEWER PROJECT STATUS

Sergei Chevtsov, SLAC, Menlo Park, USA

Abstract

This paper describes EPICS ArchiveViewer [1], a software application that presents archived process variable (PV) data in various forms. ArchiveViewer is easy to install, capable of multi-tasking, highly modular, and pluggable in many ways. It is written in pure Java and currently released in version 1.2.

The next major release, ArchiveViewer 2.0, is planned as an Eclipse [2] Rich Client.

INTRODUCTION

After EPICS ChannelArchiver [3] became the de facto standard application for archiving EPICS PV data in 2003, we decided to give the EPICS community a friendly way to view and analyze the stored data. When an extension of the popular real-time EPICS data plotter StripTool [4] proved hard to maintain, we created a new application, EPICS ArchiveViewer, from scratch.

ArchiveViewer is widely used by control system developers, physicists, and operators around the world. The application has been continuously updated to implement new user requirements.

SYSTEM DESIGN

ArchiveViewer consists of three major layers:

- "Archiver Client Adapter" layer is in charge for communication with pluggable clients of archive data servers.
- "ArchiveViewer Base" is the centerpiece of ArchiveViewer. It is responsible for processing user queries to retrieve data and present it in a plot or a spreadsheet.
- "Data Presentation Adapter" layer abstracts from concrete implementations of data presentation plugins.

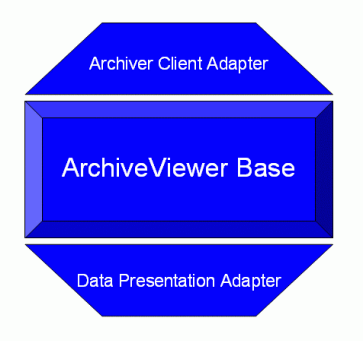


Figure 1: ArchiveViewer architecture overview.

A user requests archived data by specifying PV names (with help of regular or glob expressions) and time ranges (absolute or relative). Data is retrieved from a remote data server and cached locally. User configures plot color, type (e.g. scatter), width, and axes. Multiple domain and range axes are supported. A plot image can be printed out, or saved to a file. Plot data can be exported to a spreadsheet.

While data is processed, user receives real-time status reports and, in case of errors, comprehensible exception messages.

ArchiveViewer includes a small math library for simple data analysis and manipulations. It includes basic arithmetical, trigonometric, and Boolean functions as well as the aggregate functions.

An ArchiveViewer configuration can be saved as an XML document.

USER INTERFACES

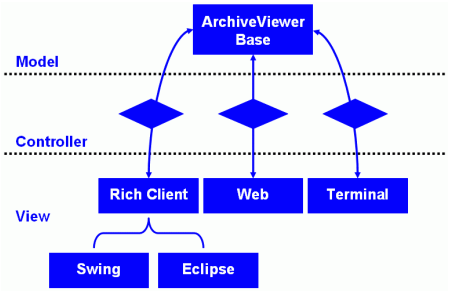


Figure 2: ArchiveViewer user interfaces.

- Terminal (command line) interface is particularly suitable for scripting.
- Web interface requires no installation on the user's side. ArchiveViewer provides a custom JSP tag library to a developer who wishes to extend the standard web pages.
- Swing front-end is very sophisticated, well documented [5], and contains an online help guide. It features GUI preferences and a wrapper for manipulating plots (incl. hooks for scrolling and zooming in/out). Swing version is available via Java Web Start, greatly facilitating installation and configuration management.
- An Eclipse Rich Client is under development. It is going to keep features of the current Swing interface and add some new ones that come with Eclipse platform.

PLUGINS

EPICS ChannelArchiver client

Development of a stable XML-RPC client plugin for ChannelArchiver data server is a driving force for

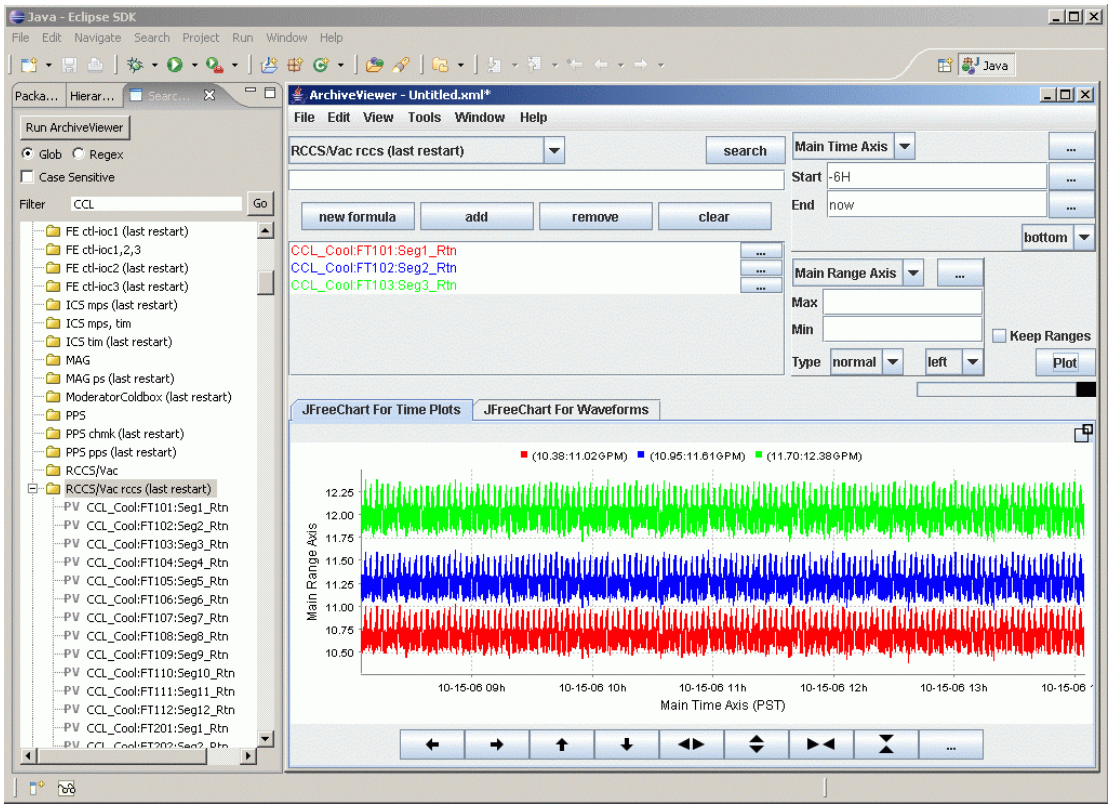


Figure 3: ArchiveViewer 2.0 Eclipse interface (under development).

ArchiveViewer. Since ChannelArchiver stores PV data in different directories, ArchiveViewer supports directories by default. As of now, this leads to some limitations concerning data interpolation and alignment. For instance, formulas can be only constructed from PV data that are stored in the same directory.

Export Plugin

So far, only the comma-separated values (CVS) format is fully supported. However, data in this format can be imported into various spreadsheet applications, such as MS Excel. After converting archived data into their string representations, ArchiveViewer displays the text on a screen. User can save it later to a file.

The CSV plugin only supports single time range requests, but can handle PV data of different types simultaneously.

Plot Plugins

Plot plugins are the major contributors to the success of ArchiveViewer.

The basic plugin can plot sets of PV data against multiple time and range axes. Range axes can be configured to feature logarithmic scales. If meaningful numeric values can not be assigned to data samples (e.g. if data status information indicates invalidity), they are drawn as clickable artifacts underneath the main graph. Plots can be resized, saved to a file as an image, and/or printed.

ArchiveViewer has a plot plugin for waveform data. This plugin acts like a virtual oscilloscope, capable of playing videos of archived data with various speeds. Multiple time and range axes are supported, too.

As of now, the final and most important plot plugin is the “correlator”. This plugin aligns scalar data samples in time and displays them in correlation to each other. One PV must be assigned to a domain axis, with data from other PVs plotted over it.

VERSION 2.0 OUTLOOK

ArchiveViewer is stably released in version 1.2. Current development focuses on the integration into Eclipse framework as part of the Control Systems Studio [6] (see Fig. 3). Additional goals include:

- runtime deployment of archiver clients (currently, clients are deployed when the application is compiled)
- an initial set-up guide
- extension of the math library
- support for German language

The development of plot plugins is highly encouraged.

ACKNOWLEDGEMENTS

I would like to thank following people for contributing to the project in various stages: Bob Dalesio, Kay Kasemir, Hamid Shoaee, Matthias Clausen, Craig McChesney, Ernest Williams, and Dave Gurd.

REFERENCES

[1] EPICS ArchiveViewer main web site, http://ics-web1.sns.ornl.gov/archive/viewer/

[2] Eclipse project main web site, http://www.eclipse.org/

[3] EPICS ChannelArchiver main web site, http://ics-web1.sns.ornl.gov/~kasemir/archiver/index.html

[4] StripTool EPICS web site, http://www.aps.anl.gov/epics/extensions/StripTool/index.php

[5] EPICS ArchiveViewer user manual, http://ics-web1.sns.ornl.gov/archive/viewer/files/manual.pdf

[6] Control System Studio main web site, http://css.desy.de/content/index_eng.html

DEVICE ADDRESS REDIRECTION AS A TOOL IN THE TINE CONTROL SYSTEM

Steve Herb, Philip Duval, DESY, Hamburg, Germany

Abstract

Naming and Name Resolution are non-trivial parts of Control System definition and implementation. A few of the problematic aspects are: 1) It is often necessary to integrate devices from neighboring control systems with different naming conventions, 2) The functionality of single devices may be scattered over multiple servers, for example history records or calibration data may be stored on separate dedicated servers, and 3) Devices which properly belong to a group are often scattered over multiple servers, with limited group functionality implemented by naming tricks or client level lists. In large control systems it is desirable that the group concepts are implemented in the middle layer software, since group functionality should exist as a Control System-wide facility, rather than as a subprocess of a single console client.

Our solutions to many of these problems include an 'address redirection' mechanism whereby the initial resolution is provided by the name server, but finer-grained resolution can be subcontracted to the device servers, which then redirect the calls to other servers, as appropriate. This is transparent to the user and maintains the efficiency of our publish-subscribe mechanism. We describe the implementation within TINE together with some representative applications.

NAMING AND NAME RESOLUTION

A naming system arranges the control system objects into sets of hierarchical families. Some possible directions for naming of control system devices, together with associated problems, are:

- A control computer oriented hierarchy of devices, tasks, and server machines: this is easy to define, and has been the basis for TINE naming, but often does not fit so well to the accelerator functionality as understood by the operations staff.
- A clever ordering of ascii characters encoding information for wild-card operations: wild-carding as a means of grouping for device operations is a bad idea when taken past some minimum level. It leads to more and more constraints on the device names, and freezes in particular views of the control system.
- An accelerator function oriented hierarchy 'easily' understood by users: for a large distributed control system similar devices, as well as the properties of a single device, may be spread over many server platforms, so that name resolution must operate at a very detailed level.

A basic aspect of hierarchies is that any structure which is a good fit to some system views will be a poor fit to some others. For large control systems, various views are needed for the different accelerator operations, and no one naming system will support all required groupings of devices. So flexibility is required in any case, meaning that the control system should have additional mechanisms to support multiple options for grouping of the objects. We describe here one such mechanism in the TINE control system [1], and some of the ways in which it is being used.

DEVICE ADDRESS REDIRECTION

How it works

Redirection is a mechanism which permits a server task to redirect selected incoming calls to another server task. The logic is shown in Fig. 1 and includes these steps:

- the client sends out a 'new' read or set request.
- the client stack passes the address string to the nameserver.
- the name server returns an IP Address and task name ('QUAD').
- the client stack sends out the call to 'QUAD'.
- 'QUAD' receives the call and checks the string against its redirection rules.
- if the rules point to another task, such as 'ABC', 'QUAD' sends this information back to the client.
- the client receives the information and passes it to the name server.
- the client receives the new address from the name server and resends the call to 'ABC'.
- this 'real' address is cached at the client level.
- 'ABC' fulfills the request and sends the return information to the client.
- subsequent calls can now be performed without these multiple bounces.

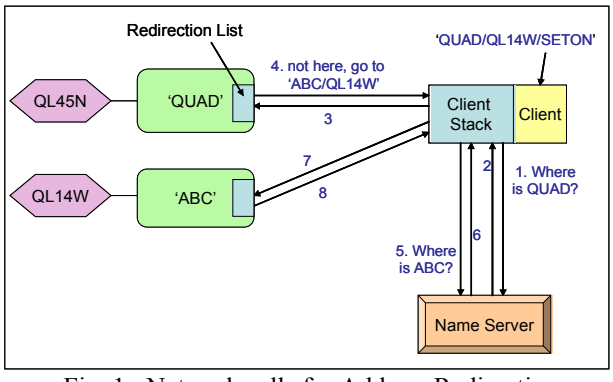


Fig. 1: Network calls for Address Redirection

The end result is that the task 'QUAD' functions as a 'virtual server' for some set of distributed quadrupoles.

The process is transparent to the 'client user' and is implemented for both the synchronous call and the publish-subscribe features of TINE. The mechanism was originally developed to support device properties spread over multiple server tasks; the result is that to a control system user it appears that all functionality for a device is concentrated at one address, which is a simplification both for the users and for the name server logic. A typical use of the mechanism has been to redirect calls for archived history data to a dedicated archive server. Another use, described in more detail below, has been to create a virtual server task for a group of magnet power supply controllers spanning multiple server platforms.

Centralized vs. Distributed Name Resolution

A possible objection might be that the full addressing information should be stored in one central location, the name server, rather than distributed at different levels in the control system. We have several remarks on this point:

First, for a system such as TINE, in which the name server entries are dynamically created during the initialization of server tasks, and for which redirection is possible both at the device and at the property level, the name server must either store entries for all possible permutations, or must implement non-trivial dynamic logic for deriving the level at which a particular string can be truncated for purposes of name resolution.

Second, having the redirection information resident on a particular server task, which then serves as the intial target for calls, could be considered a useful form of encapsulation, and as simpler than using only central storage.

Group Equipment Name Server

This is a recent addition to TINE, implemented as a middle layer server. It hosts an entire set of virtual server tasks, each consisting of a group name and a list of group members with their redirection addresses. The first time that each member is addressed, the call will pass through this server; subsequent calls go directly to the 'real' member address.

An additional feature is that server tasks may on startup automatically register their devices in one of the virtual groups, and in fact the first to register for a not yet existing group will create the group. It is not clear that this free-for-all registration is always desirable.

DEVICE GROUPS

Passive and Active Groups

The above examples regard groups as lists of devices bundled together for some operation. To the extent that this operation is just 'reads' or 'sets' using the same device properties on each member of the list, the group is not itself a 'device' in the sense of being an active control system object. There is however also a need for active groups with both client and server functionality which include their own coded methods. These methods can implement 'business logic' for the control system. Our experience with the HERA and FLASH control systems has been that all sorts of special logic is required for various combinations of magnet Power Supply controllers (PS) together with assorted peripheral switches etc.

At least for large control systems it is extremely desirable to move these methods into the middle layer of the system; too much such business logic in either the console or the front-end layers severely reduces system maintainability. Clearly the redirection does not address this need; we have nonetheless found that it can fit in well as one component of a device/group complex.

Magnet Power Supply Control Servers

Several aspects of the group implementations are illustrated by the servers for magnet PS control for the HERA and the FLASH (previously TTF2) accelerators at DESY. The HERA system consists of about 1400 magnet PS controller systems of at least 13 distinct types, addressed via 'Sedac', a DESY legacy serial fieldbus. It was very early decided to implement the control via a device server task supporting the 1400 PS instances, and a group server task supporting explicit groups with operations involving more than a single PS. These front-end and middle-layer tasks run on a single 2-processor Sun computer. The complete logical separation of the device and group functionality has been extremely useful for shielding both the console clients and the front-end tasks from the often unpleasantly complex logic of the group operations.

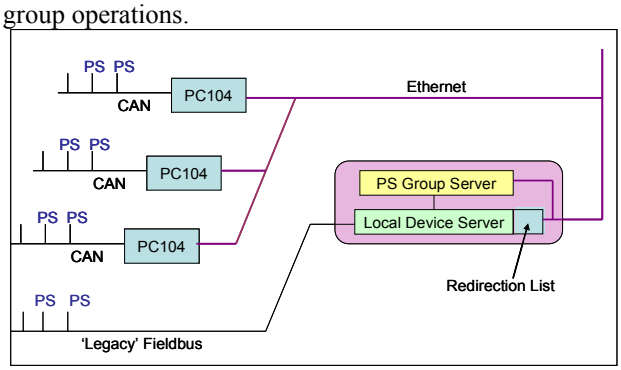


Fig. 2. One computer serves as a 'single access point' for distributed Power Supply controllers and PSC groups incorporating 'business logic' for the FLASH linac control system.

The PS control system for the FLASH linac is based on the HERA system. An additional complication is that the system is a mixture of the legacy PSs, and new PSs addressed via CAN bus. Partly as an experiment in moving toward distributed systems, the CAN busses are not driven directly by the Sun computer, but by seven PC104 modules running under Linux a slightly modified version of the Sun code and independently accessible over the network. Using redirection based on the Sun device server, all PSs appear to the central control system to be resident on the Sun, which thus becomes a (half) virtual server representing the entire system (Fig. 2)

In fact, the group functionality extends somewhat beyond simple redirection. All PS instances are represented on the Sun, either as fully functional devices, or as proxies with storage but without full method implementations. The redirection is accomplished in that each instance includes as an attribute the name of its 'home' task. The group server functions as before, except that group operations may now contain a mix of local and remote devices. Finally, the group server maintains client subscriptions to the devices on the satellite servers, and writes the results into the proxy storage. The result is that the group server on the Sun has access to current state data for both the local and the remote PSs.

'Half-Virtual' Servers' as a middle layer tool

The above construction may seem forced, but our claim is that it points the way to a possible architectural element in large distributed control systems, which we might call a 'single point of access server', namely a combination of tasks which provides a central contact point for a group of similar devices spread over the network, i.e.

- a single point of contact for the devices, as seen from the client perspective
- a preferred location for implementing groups and performing group operations related to the devices
- a preferred location to calculate and maintain group status information.

For large control systems, it is desirable to move as much system logic as possible into the middle-layer, where it is visible to all control system participants (rather than to a single console client) and to perform preprocessing on status information, so that the clients may (normally) work with summary views of the system, rather than having to piece it together themselves from atomic data transfers from each device. This model helps in that direction.

REFERENCE

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

WEB GUIS FOR THE TANGO CONTROL SYSTEM

L. Zambon, M. Lonza, Sincrotrone Trieste, Trieste, Italy

Abstract

Interactive and scalable web GUIs based on PHP have been developed at Elettra for the Tango control system. They consist of a generic control system web client and of an interface to the Tango historical archiver. Security procedures against the risk of DoS (Denial of Service) attacks and tools to easily build new pages and export data in standard format have been developed. An Ajax interface has also been implemented in order to increase the interactivity of the web interface without consuming too much bandwidth and with no interference with commands sent from the client.

INTRODUCTION

Tango is a multi-platform object-oriented CORBA powered control system software [1]. It is the result of a collaboration between four synchrotron radiation laboratories in Europe (ESRF, Soleil, Elettra and Alba), which adopted Tango to control accelerators and beamlines.

Besides the GUIs (Graphical User Interfaces) written in Java, C++ and Python using graphical tools included in the Tango package, web clients allow to access the control system and the archived data from any PC connected to the internet and running a normal web browser with no need of any plug-in.

"Canone" is a graphical animated web interface to the Tango control system. It is built mainly in PHP (PHP Hypertext Processor) [2] and animated with AJAX (Asynchronous JavaScript and XML) [3]. The external appearance is a set of "panels" from which a user can interact with the control system via web. Each panel is composed of a certain number of "widgets" and some HTML tags.

E-Giga (Electronic Graphical Interface for Global Archiving) is a graphical interface to the Tango historical archiving system.

CANONE

Architecture

A web interface is expected not to interfere with the core of the control system (a downtime due to the web is unacceptable) and, on the other hand, is expected to be as vividly interactive as possible. All interactions of Canone with the control system are performed through a tiny script in Python which acts both as Tango client using the PyTango bindings and as TCP/IP socket server (Figure 1).

A PHP front-end acts as a socket client and as a buffer layer. A multi-channel implementation has been adopted to distribute the traffic among several socket servers. In order to limit the total number of Tango calls sent to the control system all read requests and commands are stored in a buffer implemented using a tiny database (SQLite or MySQL). If the requested data is already present in the buffer and it is not older than three seconds it is retrieved from the database, otherwise a read call to control system is done and the value contained in the buffer is adjourned. With a similar mechanism multiple commands of the same type are also filtered.

The Canone core assembles the front-end, the widget library, the user administration library and other tools and glues them in a PHP powered application.

The front-end can also be used by a Simple Object Access Protocol (SOAP) server to access the Tango control system as a web service.

The installation requires a web server with little more than a basic LAMP (Linux Apache MySQL PHP) installation. The client side requires only a web browser with JavaScript and pop-ups enabled. Firefox is the preferred browser but tests have been done with Internet Explorer, Opera and Safari.

Thanks to the modular architecture of Canone it is possible, by opportunely adapting the front-end, to connect it to another control system, e.g. EPICS.

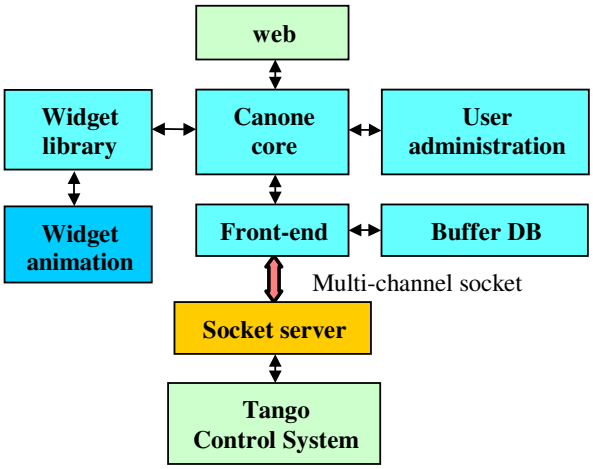


Figure 1: Block diagram of Canone

Panels

Canone allows building and customizing control panels. A simple panel is composed by a title and a table of widgets, but the page deployment may be as complex as allowed by HTML. Sub-tables and external links are examples of features that can be added.

A panel can be built writing an independent PHP script which utilizes some of the basic Canone libraries (widget, front-end, user administration, etc.). Alternatively, a panel can be easily created and modified using a web browser and then saved as a XML file on the server side.

Widgets

A widget is a graphic element that represents either a variable to display or a command.

Each widget is contained in its own class which extends the generic "widget" class. A fundamental attribute of this class is called "config" and contains all the parameters that can be customized. For each parameter there is a data structure composed by an identifier, a type definition, the default value, a short description and a long description (intended for the on-line help). The *setParam()* method allows to set all the parameters with a single string. Some widgets are implemented as PNG images others as tables, in both cases the HTML code necessary to visualize the widget is returned by the *plot()* method.

Each widget of a given panel is associated to a control system variable or command. A form allows to customize each parameter of the widget (Figure 2) including colours by means of a selection pop-up. The bandwidth utilized to transfer the widget initialization has been significantly reduced utilizing JSON (JavaScript Object Notation).

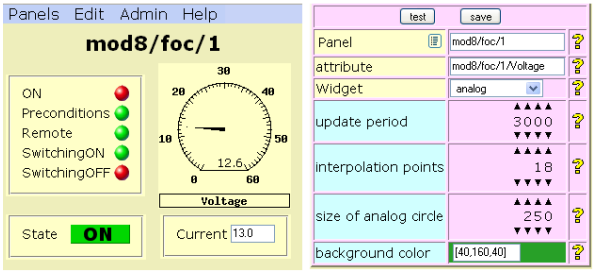


Figure 2: an example of Canone panel and widget configuration table.

Widget animation

A basic characteristic of a widget is how often it is refreshed, but for most widgets only a small part has to be changed, as all the graduated scales and graphical ornaments can remain in background. AJAX is a framework which allows the asynchronous transmission of tiny packets of data in a highly efficient way.

JavaScript allows the dynamic update of only a fraction of a web page while all the rest remains unchanged. This is easy for all widget whose animated part is in text format, but it is more complex for the fully graphical widgets. The adopted solution borrows a generic library by Walter Zorn [4] which utilizes the background colour of floating DIVs, a HTML feature supported by all modern browsers. Despite a very smart optimization has been done to improve the efficiency of the graphical library, it is important to limit the number of animated graphical elements.

All background elements are built on the server side by a PHP routine, sent as an image to the client and never refreshed.

On the client side, the browser depicts right over the background image only the minimum graphical elements got through the AJAX mechanism (Figure 3)

To save further bandwidth and improve the animation vividness, a configurable number of interpolated values between two consecutive readings can be inserted. In this way, a panel can have a refresh rate of up to 20 frames per second.

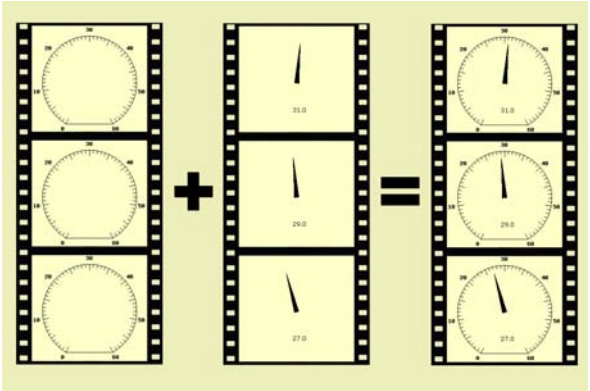


Figure 3: Widget animation.

User Administration

The user authentication process in a web environment must be robust and cannot assume at any time the loyalty of users.

In Canone, the access is validated using the information contained in a database (SQLite or MySQL). Users must be registered in the system with *username* and *password*. Each user must be part of a user group, permissions are granted to user groups. In addition, several IP numbers (and netmasks) may be associated to a user group and receive the same rights. The rights granted to a user through its *username* and *password* override the ones associated to the IP number.

There are three levels of permissions:

- *read*: only read operations are granted, any write operation is denied
- *operator*: both read and write operations are granted
- *expert*: both read and write operations are granted, panels can be modified.
  There is also a special group called "admin" with administration permissions to create and delete users and grant or revoke permissions.

The user administration utility is composed by three tools:

- *access control:* grants permissions to user groups over panels and controlled devices
- *users management*: creates, searches, modifies and deletes user accounts
- *database*: is a generic database graphical client to change any configuration. Only expert administrators should use it.

E-GIGA

E-Giga is a web interface that displays the values collected by the Tango archiving system by means of plots. It is mainly built in PHP with some parts in JavaScript and dynamic HTML. Data are retrieved directly from the archive database with the use of a set of queries. Plots are built using a native PHP library, called JPGraph [5].

The variables (Tango device attributes) can be selected from a list of attribute names together with the time period to be displayed. Figure 4 shows an example of E-Giga plots.

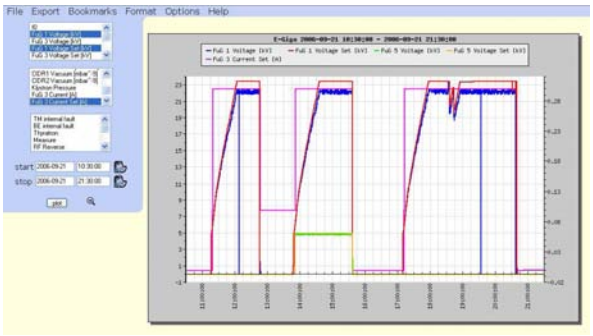


Figure 4: E-Giga screen-shot.

The vertical axis scale can be configured changing the limits (fixed, auto-scale or auto-scale with upper limitation) and the linearity (linear or logarithmic). There can be a second vertical axis configured independently from the first one.

The horizontal axis can be assigned start and stop date/times or a relative period (e.g. last 24 hours) by using a calendar and changed by just clicking over the plot (zoom in). Time scale labels on the horizontal axis are carefully placed at rounded intervals. The plot size can be freely modified. It is also possible to display a table containing the acquired data in numerical format.

A correlation plot can be obtained by placing one variable on the horizontal and another on the vertical axis. It is possible to combine variables with complex formulae by using a pop-up to write the formula and visualize it in a graphical way using MathML. Some basic statistics can also be performed.

Data can be exported in a number of external formats: PDF with graphs and tables, Excel, Matlab and CSV. Graphs can be also sent via e-mail. All configurations can be saved locally in the cookies or on the server side and easily recalled.

CONCLUSIONS

Canone and E-Giga, the web interfaces of the Tango control system and historical archiver, satisfy most of the needed requirements. They are still growing with new tools and features, but the basic functionalities are close to the maturity phase.

The whole source code, demos and images can be downloaded from www.elettra.trieste.it/~tango/Canone and at www.elettra.trieste.it/~tango/E-Giga.

REFERENCES

[1] *A. Goetz et al., "TANGO a CORBA Based Control System", ICALEPCS2003, Gyeongju, October 2003*
[2] http://www.php.net
[3] D. Crane, E. Pascarello and D. James, “Ajax in action”, Manning, October 2005
[4] http://www.walterzorn.com
[5] http://www.aditus.nu/jpgraph

A communication protocol for a distributed control system with LabVIEW

L. Catani*, INFN-Roma Tor Vergata, Roma, Italy

Abstract

Control Systems for accelerators at INFN-LNF (Laboratori Nazionali di Frascati of INFN) are mainly based on LabVIEW. VME crates hosting I/O controllers are interconnected using bus extenders translating local memory into a global shared memory. Mailboxes on shared memory are then used for communications between distributed processors. While the development of control systems for new accelerators under construction at INFN-LNF [1] should be based on well-established and more flexible technologies for communication, i.e. network, re-use of part of instrument drivers, sub-system controls and measurement applications already developed must be guaranteed. This paper presents the development of an RPC-like communication protocol based on the TCP/IP and XML tools provided by LabVIEW. It extends the features of these built-in libraries, including the managements of large binaries, and incorporates solutions that might provide compatibility with well established communication protocol, e.g. XML-RPC, while preserving full compatibility with different platforms supported by LabVIEW.

INTRODUCTION

LabVIEW is a very common development environment for controls. Limited size (and man power) projects, especially, take advantage of its ease of use and profit from the large number of tools and libraries to interface and control instrumentation, develop analysis program and display results. When the experiment or the apparatus became more complex and larger in size one might need to engage more than one computer to distribute among the different PCs the control of various components. For this purpose the before mentioned use of bus extenders is a possible option. More common nowadays is the interconnection via ethernet networks. Also in this case LabVIEW offers a number of tools to implement transfer, via network, of data between distributed components of the control/acquisition system: DataSocket, VI Server, VI reference, TCP/IP and UDP and interface to .NET and ActiveX. All above mentioned communication tools are powerful and well suited for many applications but they are not flexible enough to allow implementation of a real communication protocol. Moreover most of them are proprietary and work only between LabVIEW applications. The Internet Toolkit includes a LabVIEV HTTP server and the possibility to define the Virtual Instruments as CGI one can invoke using the HTTP protocol. This is a very general service but doesn't offer enough flexibility. LabVIEW also includes TCP/IP and UDP LabVIEW libraries providing basic tools for TCP and UDP data transmission over ethernet. They are compatible with standard socket communication being the basis for many communication protocols.

THE XMLvRPC PROTOCOL

XML (eXtensible Markup Language) is becoming a very popular way of coding data especially when interoperability and compatibility between platforms and programming languages is an issue. Communication protocols based on this coding exist, among these one of the more interesting is XML-RPC [3]. It's basically a remote procedure call that uses HTTP as the transport, or other TCP/IP and UDP protocols, and XML as the coding allowing complex, and relatively large, data structures to be transmitted, processed and returned. The implementation of XML-RPC communication protocol in LabVIEW, aimed to an accelerator control system, poses two main problems. First of all the XML code generated by LabVIEW tools is not compatible with the specifications of XML-RPC. Secondly, standard XML coding of binary arrays results in a significant increase of data size that makes the XML coding of large binaries, raw images for instance, impracticable.

LabVIEW provides a convenient set of tools to convert its data type to XML format according to the LabVIEW XML schema. Unfortunately, the LabVIEW XML schema, *LVXMLSchema.xsd*, cannot be customized or replaced by users but, if we are only interested in LabVIEW-based distributed control/acquisition systems, this is not a relevant limitation. In this case, actually, its worth to preserve full LabVIEW compatibility to take advantage of its XML library while developing a workaround for the problem of coding binary arrays efficiently.

The solution proposed in a previous paper [2] consists in pre-processing of the LabVIEW data structure before it is sent to the XML coding tool in such a way that all binary arrays are replaced by the correspondent flattened string, i.e. the ASCII string made with the same sequence of bytes as the binary array. Practically this corresponds to a type-cast of the array into a string. Because strings are copied into the XML structure without any modification, the above mentioned type conversion avoid the increase of data size consequence of standard XML coding of binary arrays. Because the pre-processor, similarly to the XML coding tool, must be able to accept all possible types of data structures as input, we first convert the data structures to LabVIEW Variants. The Any-to-Variant function converts any LabVIEW data to a format that can be manipulated independently of the original data type. A Variant can be unpacked, its content modified (adding, deleting or replac-

* luciano.catani@roma2.infn.it

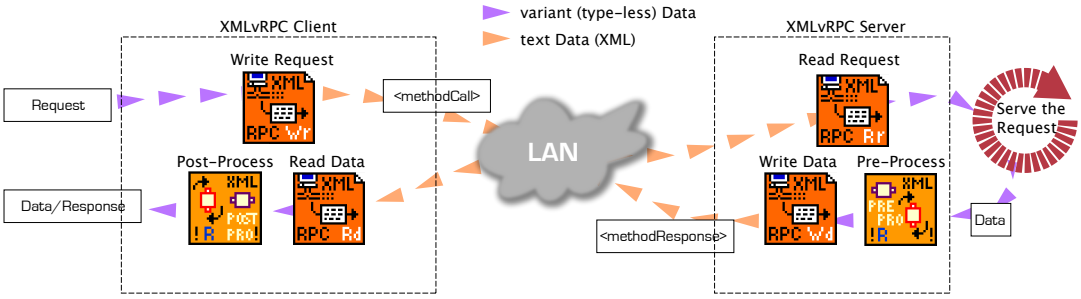


Figure 1: XMLvRPC client and server communication

ing data, for instance) and then converted back to a "standard" LabVIEW data (numeric, text, array, cluster, etc.). The pre-processor developed for this application is a VI that recursively (its "Reentrant execution" option must be checked) searches for nested binary arrays into a LabVIEW data structure converted to a Variant and replace them with the correspondent flattened strings. The processed data is then coded into a XML string and reduction in size, with respect to the non-pre-processed version, is clearly significant. If we compare, for instance, XML coding of a data structure (e.g. a LabVIEW cluster) that includes a 640x480 2D array of unsigned-bytes , a typical pixels map of a CCD camera, reduction in size obtained with the pre-processing described can be a factor 100 or more.

It must be noted that when a binary array is flattened to a string, some relevant information about the original array is lost. As consequence reconstruction of a binary array on the receiver side is not possible unless we supply, by other means, the dimension(s) of the array and its data type and size (the number of elements for each dimension is included by LabVIEW in a header of the flattened string).

The solution that has been chosen is very simple: the missing information, the dimensions of the array and its data type, properly coded and formatted is appended to the name of the variable. As an example, the variable 'image', being the 640x480 2D binary array previously mentioned, after the pre-processing procedure transforming it into a string will change its name into 'image_2_U8'. On the receiver side a post-processor parses the LabVIEW Variant obtained converting the XML data. It selects the strings that it recognizes, by their names, as flattened binary array and unflatten them into an array having the indicated dimensions and data type. Once the XML coding is defined the implementation of the XML-RPC-like communication protocol, that I called XMLvRPC, is straightforward. See Fig.1.

DISTRIBUTED CONTROLS WITH XMLvRPC

CLIENT AND SERVER

The core of the XMLvRPC protocol are the XMLvRPC_Server.vi and XMLvRPC_Client.vi whose components are shown in Fig.1. Since data is passed to/from these VIs as Variant, that is a type-less data, the XMLvRPC Client/Server VIs can present a common interface to all calling VIs, yet compatible with any type of data they need to transfer across network.

PRE and POST processors take care of large binaries: binary arrays are flattened (type-cast) into strings and then coded into XML reducing the size of the coded data structure. 3+3 symmetric VIs have been developed to implement the client/server protocol. On the client side the XMLvRPC_Write_request.vi initialize the query to the server. The calling user application must provide the server address, the method name and optional parameter. The server is continuously listening on the predefined TCP/IP port. As soon as a client opens the connection it uses the XMLvRPC_Read_request.vi to read the methodCall. Then it runs locally the VI that serves the method (*method*.vi) and the variant data produced as result is passed to the pre-processor XML_preR-processor.vi to search for binary arrays.

It must be noted that since data is passed to *method*.vi as variant the latter, independently on the method they serve, have the same TypeDef (practically they have the same input/output parameters) and thus can be programmatically loaded at run-time and executed, provided their name corresponds to the method name. It also means that when a new method is added to a server (similarly on a client) the server source-code doesn't need to be modified to include the call to this new VI. It will be sufficient to copy the VI that serves this new method to the directory where the server XMLvRPC_server.vi searches for *method*.vi that serves the methodCall it receives from clients.

After pre-processing the server uses the XMLvRPC_Write_data.vi to send the methodResponse to the client that is waiting for the result of its call. The latter uses XMLvRPC_Read_data.vi to receive the methodCall from the server. Then it runs the post-processor XML_postR-processor.vi to convert the flattened binaries, if any, back into the original arrays. Data received are then given to the calling user application or to another VI defined by the name of the methodResponse. It will be loaded at run-time and executed similarly to the server side. It means that the protocol also support asymmetric methodCall/Response: on client side the *method*.vi that

displays or analyze the data received from the server can be different from the one that originated the methodCall. The methodResponse can indicate another method serving the response on the client, according to the data produced from the *method*.vi on the server side, if it can have different forms.
Possible future compatibility with standard XML-RPC since data is passed in XML

COMPONENTS IN A XMLvRPC DISTRIBUTED CONTROL SYSTEM

Fig.2 shows an example of components in a XMLvRPC distributed control system. Controllers run front-end applications: they are either the interface to equipment or provide general services. Consoles run user applications or analysis and measurement procedures. Consoles directly connect to Controllers to run remote procedure provided they know (the IP address of) the controller in charge for the particular I/O channel (or service) and the methods made available from it. This information is provided by the Configuration Database on request by the Console (or a generic client). The Configuration Database is thus the repository of the system configuration files collected from any controller at the time they startup and register to the system.

To summarize, TCP/IP and UDP services are the following:

XMLvRPC TCP/IP Server: It runs on each controller and on the Configuration DataBase. It serves XMLvRPC *methodCall*. For each controller, available methodCalls are those listed in the XMLvRPC_ClientServer/methods_svr directory; Elements under control are those listed in XMLvRPC_ClientServer/elements_svr directory.

XMLvRPC TCP/IP Client: It runs on each console (a client in general) and send XMLvRPC *methodCall* to XMLvRPC TCP/IP Server according to the requests of the control panel or user application.

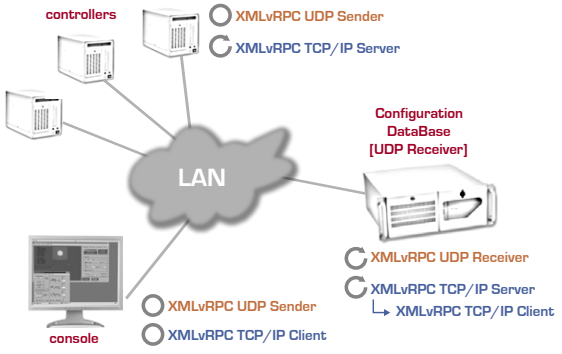


Figure 2: Typical components in a XMLvRPC based distributed control system. Main services are also shown.

XMLvRPC UDP Receiver (Configuration DataBase) It runs on the Configuration DataBase and serves *synch_me* or *register_me* methodCalls sent by controllers or *locate_cdb* sent by consoles at startup.

XMLvRPC UDP Sender: It runs on the controllers/consoles at startup. It sends *synch_me* or *register_me* methodCalls to Configuration DataBase to register the new controller in the system. Consoles use it to locate the Configuration DataBase.

Configuration DataBase: It is the repository of the configuration files and provides to the consoles information about the controller in charge for a given element.

INITIALIZATION AND REGISTRATION OF SERVICES

At startup each controller sends a UDP-broadcast to register on the Configuration DataBase by sending *synch_me* or *register_me* methodCall (Fig.3). *register_me* is used if the controller provides all its methods and elements. *synch_me* is used if some methods (and elements) are provided by the Configuration DataBase. If the system has more than one Configuration DataBase for redundancy purposes, both will receive the request to register the controller in the system. The Configuration DataBase detects the UDP-broadcast and then sends to the controller a TCP/IP *get_elements* methodCall and then a *get_element_conf* for each element listed in the previous methodResponse received from the controller.

Practically, local services (i.e. those specific for a controlled elements) are configured directly on each controller while global services (e.g. back-up, restore etc.) can be configured centrally in the Configuration Database.

Consoles and high level applications relay on the Configuration DataBase to locate the controller in charge for a particular element. They use an UDP broadcast to find the Configuration DataBase, i.e its IP address. At this point one can either decide to receive the complete configuration of the system at once and refresh it periodically or relay on the Configuration DataBase each time a client needs to identify the controller in charge for a particular I/O channel or service.

THE XMLvRPC SUITE OF VIs

Fig.4 shows the structure (directories, VIs and configuration files) of the XMLvRPC package.

The installation can be identical for any component of the XMLvRPC system. The role and the services available for each component are configured by means of the configuration file Config_local.xml and by copying the needed VIs in the methods_svr or methods_clt for a controller and a console respectively.
If the local computer runs a Configuration Database the methods to be used for this service are in methods_cdb_rec. The system_database directory contains a directory for each

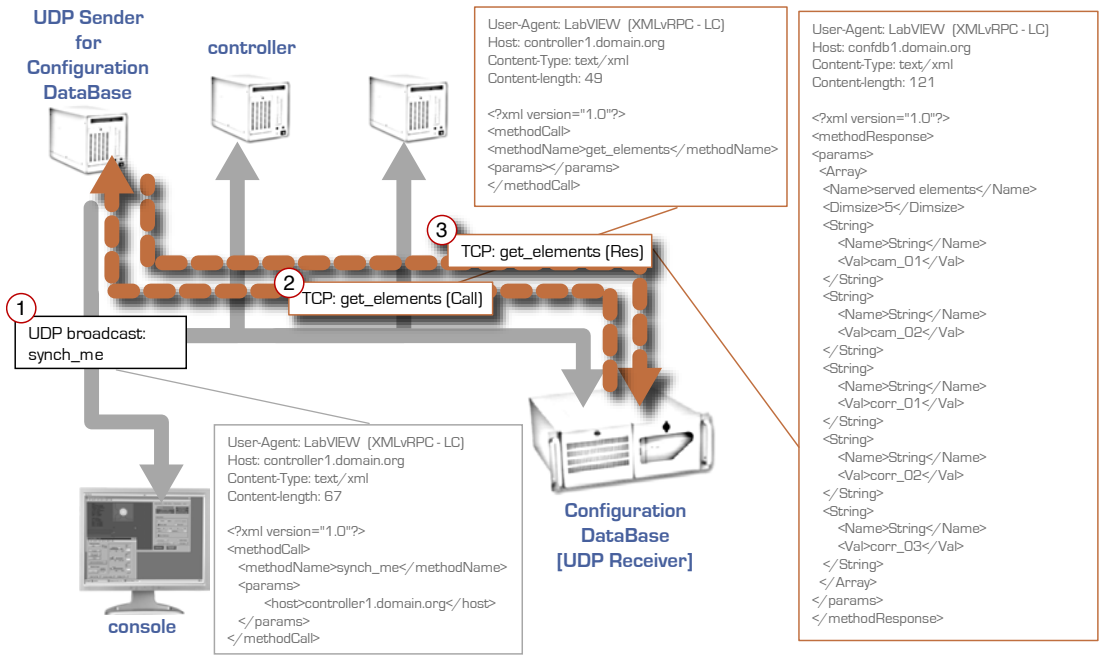


Figure 3: Synchronization of a controller with the Configuration database at startup by means of UDP-broadcast.

of the controllers (servers) that registered the system in which the configuration files of the controller elements are stored.

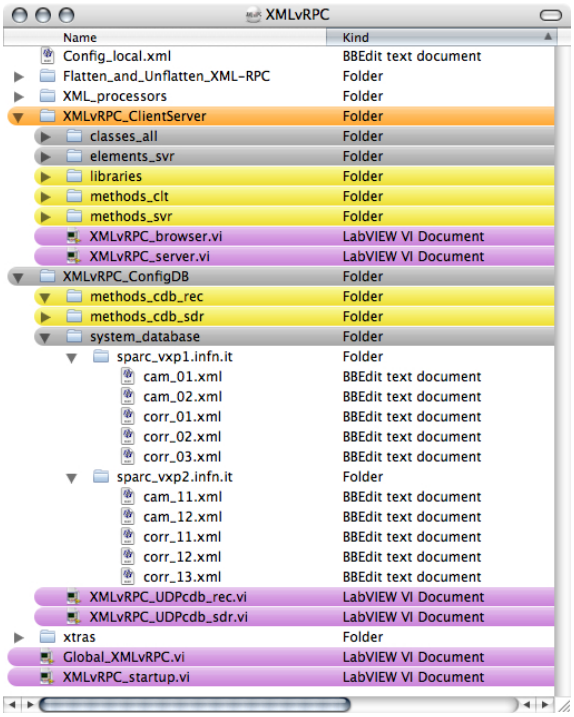


Figure 4: Directories, VIs and configuration files in the XMLvRPC package

CONCLUSION

XMLvRPC, an XML-based communication protocol for LabVIEW provides feature allowing its applications to LabVIEW-based distributed control systems. LabVIEW programming is simplified because XMLvRPC provides a general communication system that accepts all kind of data structures and relies on strings and Variants for its communication components. Although XMLvRPC application is restricted to LabVIEW-based distributed control system its structure is such to make compatible with XML-RPC with limited effort. The strategy used to manage large binaries could be useful also in that case because it doesn't introduce a violation of the XML-RPC standard. The standard XML-RPC client/server, in fact, will simply pass the string data (string data type is obviously allowed in XML-RPC) to the user application that is designed to handle the result of that particular methodResponse. The latter will contain the instructions to convert the string back into the original binary array.

REFERENCES

[1] G. Di Pirro et.al. "First operation with the SPARC Control System", these proceedings

[2] L. Catani et.al., A Large Distributed Digital Camera System for Accelerator Beam Diagnostics, Rev. Sci.Instr. 76, 073303 (2005)

[3] XML-RPC Specification - http://www.xmlrpc.com/

USER REQUIREMENTS FOR THE PETRA3 CONTROL SYSTEM AT DESY

M. Bieler, A. Brinkmann, U. Zobjack, DESY, Hamburg, Germany

Abstract

In 2007 the PETRA accelerator at DESY, Hamburg, will be converted into a high brilliances synchrotron radiation source. At that time the control system for PETRA will be upgraded. As part of the design process for this new control system members of the operations group have gathered their requirements for the new control system. Some of these requirements will be presented in this paper.

INTRODUCTION

The user requirements shown in this paper were gathered from the operations group at DESY. Good and bad examples shown here are from the DESY control room. Not all (bad) examples shown here are still in use at DESY. Most of the rules proposed here can probably be found in any software style guide (but who reads them?).

RULES FOR GOOD APPLICATIONS

Colors

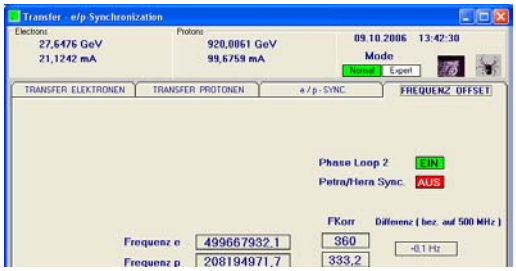
About 9% of the male population can not distinguish between red and green. Therefore, color coded information should always be accompanied by text.

How many Buttons per Square Inch?

The "All in One" style with all information and all buttons related to one topic on one application is often problematic. Such applications show too much information and to many functions in one application. Operators often have a hard time to find what they are looking for.

Using a number of "Pull down Menus" makes it difficult to find the requested information, because a search by trial and error is very time consuming.

A good tradeoff is the use of "Folders", with the information separated in folders and all relevant information visible in each folder.

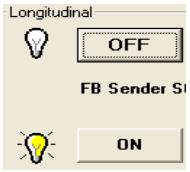


Expert's Menus

Buttons for experts are dangerous and confusing during normal operation but necessary for troubleshooting. Expert's menus should be hidden during normal operation, but easy to access in case of trouble.

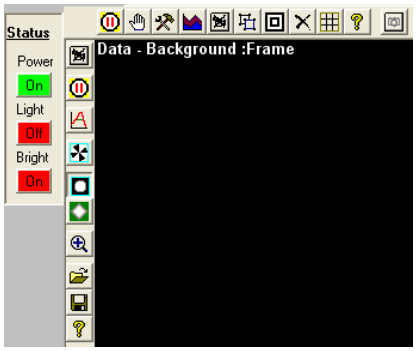
Labeling of Buttons

The label on a button should always show its function, not the status of the device attached to it. Additional information about the status of the device is welcome, but should be separated from the button.



Symbols

Symbols on buttons or sliders should only be used, if their meaning is well defined.



'OLD' Button

If parameters are frequently used for (mostly fruitless) optimization, an 'OLD' button is very useful. It sets this parameter back to the value it had when the application was started.

Size

With many applications running on one screen, the size of each application becomes a problem. Each application should be scalable according to the users needs. Scaleable fonts would be ideal.

Some applications can be of small size in normal operation, but should offer more information in an expanded version in case of a fault or for special purposes.

'Look and Feel'

Often used applications should have individual 'Looks', so that the operator can distinguish between them at a glimpse without reading the fine print. Rarely used applications (e.g. archives) should all have the same 'Look and Feel' for easy use.

'About…'

Every application should offer information about

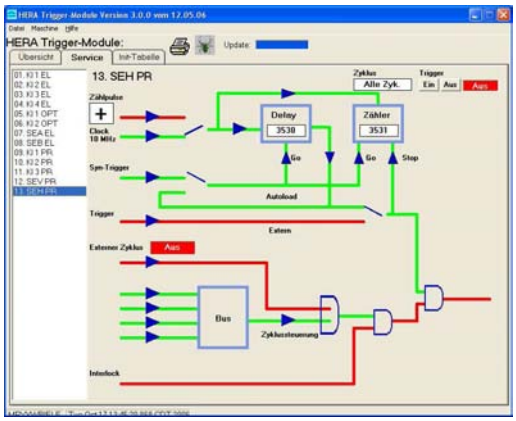
- the software version number
- the date of the last change
- locations of the involved computers
- name and phone number of the author
- name and phone number of the people responsible for the hardware driven by this application (e.g. vacuum valves)

Optimization Plot

When a parameter like experimental background is optimized, it is useful to see a plot showing the figure of merit (experimental background) versus the tuning parameter (orbit bump amplitude).

Simplified Drawings

For some applications a simplified drawing of the components involved can be helpful for the occasional user.



Knobs vs. Sliders

Wherever fine tuning is required (luminosity, background…), big round knobs are better than sliders. Turning a knob fast for three turns and than slowly for another 1/10 of a turn is easier than doing the same job with a slider.

Different applications can be attached to the same knob; a short text at the knob shows the actual function of the knob.

A counter at the knob keeps track of the changes applied.

Archives

All parameters should be archived at an individual rate. The archive reader should plot them on the same time scale. See the paper by Mark Lomperski in these proceedings.

Miscellaneous Problems

If an operation takes time (e.g. heating up a thyratron), show the timer on the screen!

Error messages should be as specific as possible ('Error 41' does not really help to cure the problem).

Cryptic abbreviations should be avoided.

Different applications should use the same word for the same function (End, Stop, Exit, Close, Back,…).

Every application should be available on every console.

No application should require a different console type.

Conclusions

The operators don't care if it is Visual Basic, Java or something else, they want good application software.

Before new application software is being implemented, the operators should be asked what they need and how they need it.

Some of the best control room applications are those made by operators.

A good way to help the controls group to understand the needs of the operators is to let the controls group do part time shift work.

OFF-LINE ANALYSIS GOES ON-LINE!

Mark Lomperski, DESY-MIN (NOT the Controls Group!), Hamburg, Germany

Abstract

With the increasing complexity of accelerators, operators need all the help thay can get to analyze the available information quickly and to make correct operational decisions. Software tools for such data analysis often land in the Off-Line Analysis category, written by and for experts and which remain separate from On-Line controls. Making these tools usable on-line requires flexible controls software, good communication between subsystem- and software-experts, and support from management to give the necessary priority and time to putting a (never completely finished) tool together. In this report the importance of this aspect of controls will be covered with selected examples from the central accelerator control room at DESY.

THIS USER'S BACKGROUND

This user is a member of the DESY Scientist Shift-Pool, a group of scientists which participates part-time together with engineers and technicians to run the accelerators at DESY. The members of the shift-pool only do shifts part-time. This fact puts extra importance on user-friendly console applications.

The continuous push for increases in performance and efficiency requires better understanding of the accelerator, which brings more and more data to the operators. Software tools are important to help analyze these data. Analysis may be needed in real-time, during operations, by the operators. This is a challenge for a control system: the search for optimum solutions for problems with machine-physics, automation, controls, software. To help the software group with these special tasks, I began to write application software.

EXAMPLES OF ON-LINE ANALYSIS

To get things started, I give a few simple examples of on-line data analysis in the DESY control room.

Beam Position Monitors (Orbit or Trajectory)

Beam position monitors are an important diagnostic in most accelerators. Not only are the absolute values of the positions of interest, but also the differences or changes with respect to reference data. These references could be measured together with the nominal data e.g. the difference of the positions of two bunches in a bunch train. The references could have been taken moments before, before a correction coil setting was changed to steer the orbit. The reference data could be a "golden orbit" saved during optimal conditions, far in the past.

Thus the measurement of changes in beam position depends on how the question is asked: namely, relative to what? The console applications must allow for easy selection of reference data types. An example of one of the orbit-displays was shown in the talk. During operations, we display absolute orbits and various difference orbits, and monitor how these differences change. Orbit corrections can be made using the console applications to correct relative differences in any of the display-modes.

Temperature Alarms

At high energy, the HERA electron beam produces large amounts of synchrotron radiation which can hit the vacuum chamber, heating it and producing leaks. During the ramping procedure, many (>150) temperatures are monitored and can automatically trigger a beam dump. These temperatures are also monitored by the shift crew, and with orbit-steering the light can be steered away from sensitive vacuum chambers, reducing temperatures. At the beginning of a ramp, the sensors have different initial values and different alarm thresholds. For the operators, the development in the temperatures is followed in various ways: absolute temperature, changes with respect to the initial value at the start of the ramp, and relative to the alarm thresholds. The display program must allow the operator to switch quickly between these different display-modes. A console application which allows this was shown in the talk.

This type of data scaling is used in the general archive viewer programs.

Proton Beam Losses

The HERA proton beam must be dumped to protect the superconducting magnets in the case that high losses are detected. The causes of these losses are not always clear: for example a trip of a technical system may not be found. Due to the long fill-time (~2 hours) time is saved by studying the losses before another fill is made.

To aid in the detective work, it is useful to analyze the time structure of the losses (over what time scale have they increased?) and the distribution around the ring (localized in one area, or spread-out?). With 300 monitors, this task is tedious done by hand. An extension of the console application for the loss monitors has been written to search through the data: the user can select time windows, and thresholds, to help select and analyze the distribution in time and space. This application, with input parameters for thresholds and time window, was shown in the talk.

This type of functionality is also common for applications which show status information. The user can select a subset of the complete information. For example, when a component trips, the status information is sorted through, and ONLY the bits which are changed are

displayed. The other status bits, which still have an OK status, do not need to be shown.

MY REQUIREMENTS FOR THE CONTROL SYSTEM

Starting with these examples, one can characterize some aspects of the console applications which are in use in the accelerator control room at DESY.

Data Flow

Data for an application can come from various sources. Live data can come from a front-end server or from a middle-layer data server which may have the task to calculate extra stuff. History Data can come from the front-end, a middle layer, or from a central-archiving system. Console applications need to be able to switch between these sources, either automatically or by command of the operator.

Data Manipulation

The console application must be written in a language to allow data manipulation. Not all number-crunching can be done in a middle layer. In the simple examples discussed above, the subtraction of orbits and the scaling of temperatures were described.

Graphical User Interface

The presentation of data analysis must be kept as clean and user friendly as possible. This is a goal of any control software. Data analysis makes console applications more complicated, and so more effort must be made to produce an application which helps guide the operators to where they want to go.

For example, applications should be able to switch easily between live data and history data – and these choices are best integrated into a SINGLE program. A decision which source of history data should be accessed is made by first checking how much data is stored by each source. If the central archive server has sufficient data, then it is collected from there. This happens without intervention by the user.

User Interface Development

The development of analysis applications never stops. New questions arise, new data is made available, or simply because the questions become clearer and the type of answers can be refined. Analysis applications require substantially more time for both discussions and for programming before the end-product reaches a state usefulness for the operators.

NEW DEVELOPMENTS IN CONTROLS?

These requirements and goals for control system applications are apparently not universal.

Quick-Applications

I have heard buzzing (by Controls-Bees) at DESY about new software "frameworks" which allow simple browsing through control-system structures, with the feature that one can "select" and "drop" these "objects" into a displayer. I apologize if my software-vocabulary is used incorrectly to describe this functionality.

The super idea, I guess, is to quickly and easily produce graphical displayers for control system data. The "user" only needs to "Drag 'n' Drop" or "Cut 'n' Paste" and WOW! One has a console application!

In the DESY control room, there are VERY FEW client programs which could be substituted with such a quick-glue-together application. The applications are "subsystem" oriented – and all required functionality for the subsystem is built into this client. Applications which ONLY do display with limited additional functionality are rare. Such a "framework" would be useful for testing, but that is about all, and can definitely not be a replacement for a framework for more complex/flexible application programming.

Expert-Programs

Something else which I have heard from Software-Types is that certain projects are for "experts" and so lie outside of the control-system.

It can be difficult to separate which applications are necessary for operations, or useful for operations in a pinch, or only required for use by experts. It is best for operations to have as many software tools as possible available for special functions. These tools should be integrated into the control system; NOT be external programs.

SUMMARY

The presentation layer is very important for the operational efficiency of an accelerator. The accelerator can have super hardware and gorgeous servers, but if the presentation layer suffers, then the efficiency suffers. The symptoms are not necessarily DOWN-TIME of the accelerator, but DEAD-TIME of the operators as they use time unnecessarily to squeeze information out of the data in order to make decisions.

Accelerators produce large amounts of data and these data need to be presented to the user in the most user-friendly way possible. Complex software is required to produce simple applications based on complex data.

With complex data, not all manipulations can be made in a middle-layer: computations must also be possible at the console level.

Sophisticated software is required to produce simple applications based on complex data. New ideas about "drag and drop" clients can be useful only for test purposes, not for final clients used for operations.

A Users Perspective

Isadoro T. Carlino, Jefferson Lab, USA

Introduction

Industrial design is not a subject that is often associated with computer programming in general and control system design in particular. Yet the principles of Industrial design have driven the design of analog controls from the beginning. From the shape and placement of breaker controls on a power distribution system panel to the array of lights on an alarm display panel the influence of industrial design in the analog world is easy to see.

In the digital world we have made the leap away from the constraints imposed by analog controls, yet the influence of the limitations and paradigms of the analog world continue to carry into the digital world. That is as it should be. Many of these principles are based on human needs, rather than analog limits.

Jefferson Lab Overview

The accelerators and support systems at Jefferson Lab (JLab) support science at three main experimental halls in the Continuous Electron Beam Accelerator Facility(CEBAF) and a number of laser experimental halls at the Free Electron Laser (FEL). A Central Helium Liquefier (CHL) provides cryogens for these facilities. The control system used is EPICS.

At Jefferson Lab control system users can be divided into distinct groups, each with their own needs. Operations staffs at the CEBAF Accelerator, CHL and FEL. Experimenters at the experimental halls. System experts in the support staff, mostly technicians and engineers. Accelerator physicists, who often require a wider view than system experts and a deeper view than operations. The needs of each group are different, and must bow to the principles of usability. This paper will concentrate on the needs of operational users.

Limitations

One of the limitations of the digital control system is that control feedback is the responsibility of the software designer. With analog controls position is often a cue to state. Sound is sometimes an indicator of change of state. All such visual/audible feedback must be deliberately implemented in a digital control system.

Perspectives

The user perspective is colored by the environment. The control room environment is complex. Unlike the system expert, who typically concentrates on a single system, device or limited set of objects, the operator must monitor a large variety of different systems.

Operators are jack-of-all-trades and not generally system experts. They also do not typically have the luxury of becoming experts in the detailed operation of all accelerator systems. Conversely they often have a greater knowledge of the interaction of systems in a real world setting than experts who are concerned with the operation of a single system.

Because of their limited knowledge of individual system operation and the fact that they often have to operate a system only in a severely limited operational envelope operator controls are often made up of a distinct subset of expert tools. At JLab this often means control screens with specific controls relevant to particular beam states or operational conditions and controls from multiple systems group together on a single panel to support a specific beam evolution.

Unlike system experts, accelerator physicists often operate devices from many systems, analyzing the responses of the accelerator monitoring devices in complex ways, with the goal of improving machine operation. Often these measurements or machine optimizations can be distilled down to a set of steps or actions which can then be performed by a less knowledgeable individual by encapsulating it to a tool which can perform the procedure. At JLab a variety of such tools are implemented; some written in C, others in C++, many in tcl/tk, and some at the EPICS ioc level, using State Notation Language, or the EPICS database. An important consideration for the designer is that from the user's perspective the language/system of the implementation is irrelevant.

Human Machine Interface

At JLab there are no less than 8,000 EDM control panels, of which 3,000 are easily accessible to the control room staff. Of this number they regularly use no more than several dozen. Many of the others will only be used by the operations control staff when there is a problem. If the colors, layout and design of these screens are not consistent the staff will have difficulty interpreting the information on them. It is quite common for a screen to be used only during accelerator optimization and steer-up, a process which happens only two or three times a year. Due to the nature of rotating shift work it is quite common for a particular operator to be required to use a screen with little or no prior exposure. Consistency of design is once again a vital component of use.

Colors

Color is a very traditional way to differentiate fields on a GUI interface. At JLab we have standardized on a cyan background for control devices. As can be seen in Figure 1:Frequency Control EDM Screen, control devices, independent of their widget type are colored cyan. Text read back fields of all types are dark blue with white writing. One of the benefits of these two color combinations is the ability to differentiate control text fields from read back text fields on a screen shot printed by a black and white printer.

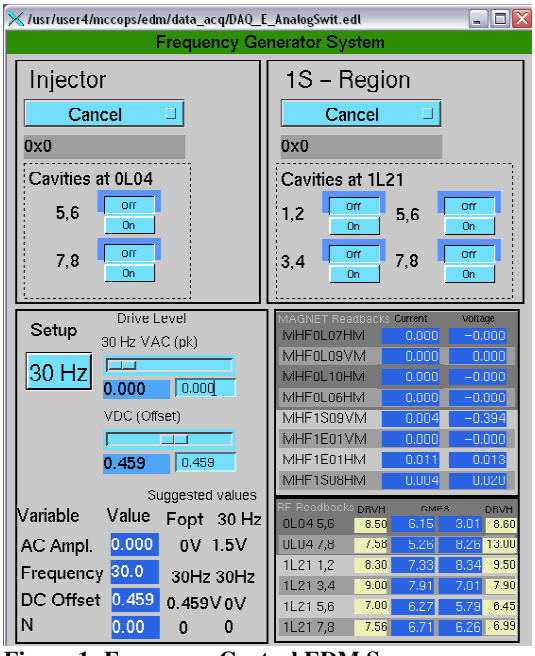


Figure 1: Frequency Control EDM Screen

Device status is most typically designated by the traditional use of red/yellow/green. Green in our case is a status which will allow accelerator operation with red indicating a problem. Yellow typically indicates when a device, such as a valve, is neither opened nor closed. The addition of text makes it possible to determine status even for those with problems differentiating color.

On expert screens we sometimes use other colors to designate controls of special importance, usually gold, but sometimes red/green. Color coding is not used exclusively on edm screens. Tcl/tk tools follow the standard pattern as can be seen from figure 2.

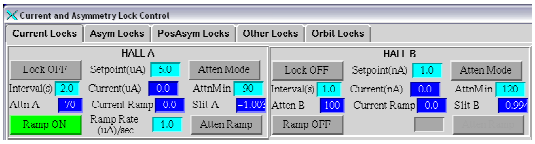


Figure 2: tcl/tk Lock Screen

Screen Sets

Screens at the CEBAF machine are made available from the Accelerator Main Menu, Monticello (figure 3). Monticello is divided into a main section, based on area/function, and a subsystems section based on system. Screens in the area/function section primarily consist of panels which display the status of devices from a specific location, or screens used to perform a specific procedure. The gold buttons open the single screen from the set which is which is used most often, typically the highest level screen of a system.

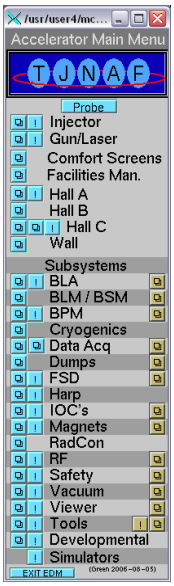
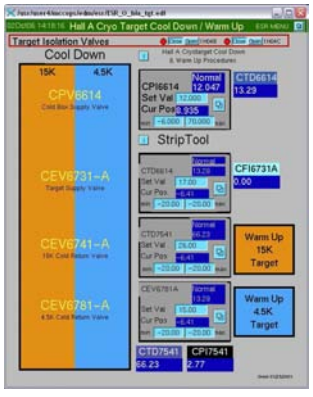


Figure 3.

The Hall A Cryotarget cool down/warm up panel (figure 4) illustrates a screen designed to perform a specific procedure. Note

Figure 4.



that devices are positioned in a logical order so that an operator following the procedure will advance down the screen. A link is available to the written procedure and a related screen button is present to open the menu leading to the approximately thirty expert screens of the systems tangential to this procedure. The operator sees only the controls and indications necessary to accomplish the task.

Widgets

It is important that widget function naturally correspond to device operation. For example, superconducting RF cavity gradients typically operate at a fixed setting. For an engineer's standpoint a simple text control field would be sufficient to set RF gradient. However, operational experience shows that cavities sometimes fail to maintain gradient, a state known as cavity SOS'ing. Operator response is to de-rate the cavity temporarily by stepping down the gradient, allowing the cavity to stabilize and then returning the cavity gradient to the original setting. A simple text field is not the best type of control to perform this operation. A slider, which has a controllable step size, is a much better tool for this task.

Complicated activation sequences should be reserved for cases where the ramifications of inadvertent operation have a high penalty. For example, accidental opening of beamline vacuum valves in the vicinity of an RF zone could result in dire problems should the system be opened for maintenance. Such valves require a confirmation from the user after the open command is given (Figure 5 SRF valve). Even in this case a combined control is provided, because in normal operation it would be onerous to require the user to confirm the operation of a dozen valves individually.

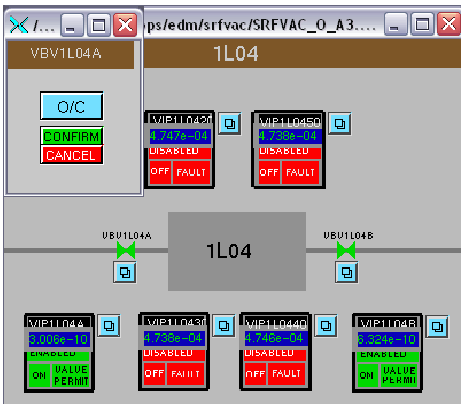
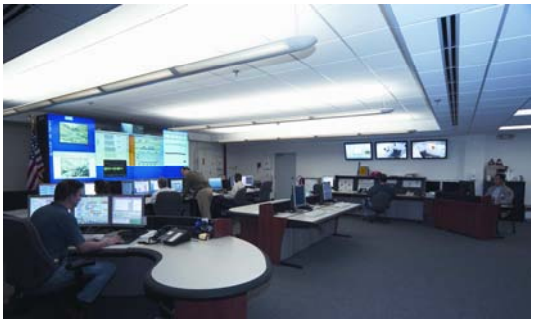


Figure 5: SRF Valve Control EDM Screen.

Screen Displays

Just as users needs are different the display environment of each group of users is different. While most users at JLab use single or double headed display monitors, users in the control room operate from triple headed monitors. They also have a 16 ft by 6 ft display screen in front of the control room, as well as 4 by 2 status display LCD and plasma monitors. Screens which would be impractical to use on a standard 21" monitor are very usable when displayed on a display wall. For example, while it is impractical to display much more than a single arc of beam position indication on a single monitor a complete beam pass of the 5 pass machine can easily be displayed on the display wall. Constraining operational users to small screens because others are so constrained is not optimal design.



Alarm Handling

Because of the complexity of the control room environment it is unrealistic to expect operations staff to monitor the status of tens or thousands of system parameters. Such monitoring is best done by an alarm handler. Many kinds of alarm system paradigms are possible. At JLab we use a central alarm handler for most systems. This standard EPICS software has both audible and visual cues when an alarm is active. The system is logically organized based on area. This replaced an older systems based alarm setup, which while it made sense to the system experts who created it, did not make sense to the operational users who actually used it.

We also use a separate system for fire, oxygen deficiency and radiation. This system, is run by a PLC which triggers an audible alarm as well as a flashing LED for each alarm type. Fire alarms are further annunciated using a flashing strobe light. An MS Windows based fire alarm monitoring system supplements this system,

giving more detail to assist in fire investigation. The screens are geographically based, showing the user the location of the alarm on a floor plan of the affected building. Once again information is display in a manner that allows a user who is only peripherally familiar with the system to use the software.

Expert Systems

An example of an application that encapsulates the knowledge of a system expert is the Injector Setup tool (figure 6.) This software allows an operator to setup the CEBAF injector, a procedure which originally required an injector expert. The tool has links to procedural help as well as help for the tool itself. The operational user moves down the screen in a logical order to complete the procedure. Buttons either perform setup actions directly or open other tools or EPICS control screens to allow individual steps to be completed. Automatic entries to the electronic logbook are made by the software, ensuring consistent entry of information.

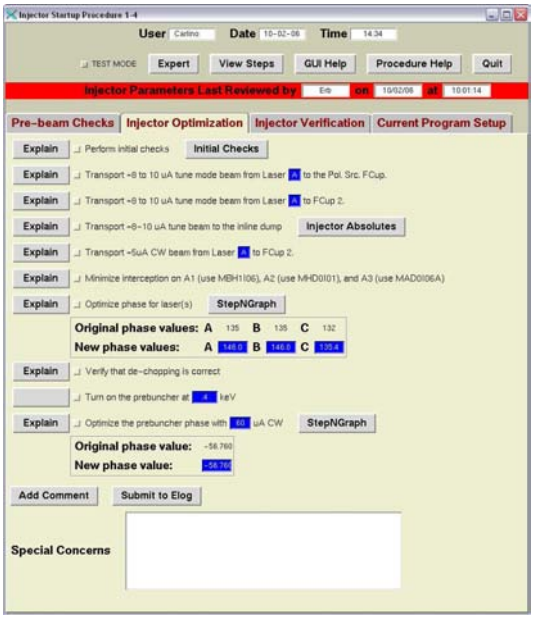


Figure 6.

From a users perspective this software embodies all of the best of user friendly design principles. The purpose of each step is explained. Tools are readily available to the user. The order in which the steps must be performed is obvious to the user. The use of this tool requires an overall knowledge of the accelerator, but does not require that the user be an expert in the Injector systems that are being configured.

Summary

The user is often the most overlooked component of control system design. For operations users the complex control room environment requires that tools be consistent. Language/System is irrelevant to the operations user. Usability is the most important factor to the operational user. The many considerations necessary include audio, visual, and Human Machine Interface parameters. These include procedural and help interfaces and logical screen design. They sometimes encompass factors outside the conventional program design paradigm to include analog annunciators. All of these factors effect the user experience.

.

A PROTOTYPE OF A BEAM STEERING ASSISTANT GUI TOOL FOR ACCELERATOR OPERATIONS

M. Bickley and P. Chevtsov, Jefferson Lab, Newport News, VA 23606, USA

Abstract

The CEBAF accelerator provides nuclear physics experiments at Jefferson Lab with high quality electron beams. Three experimental end stations can simultaneously receive the beams with different energies and intensities. For each operational mode, the accelerator setup procedures are complicated and require very careful checking of beam spot sizes and positions on multiple beam viewers. To simplify these procedures and make them reproducible, a beam steering assistant GUI tool has been created. The tool is implemented as a multi-window control screen. The screen has an interactive graphical object window, which is an overlay on top of a digitized live video image from a beam viewer. It allows a user to easily create and edit any graphical objects consisting of text, ellipses, and lines, right above the live beam viewer image and then save them in a file that is called a beam steering template. The template can show, for example, the area within which the beam must always be on the viewer. Later, this template can be loaded in the interactive graphical object window to help accelerator operators steer the beam to the specified area on the viewer.

INTRODUCTION

The CEBAF accelerator provides nuclear physics experiments at Jefferson Lab with high quality electron beams. High quality of the beam means its precise position on a target, almost ideal gaussian profile, and very small (~$2 \cdot 10^{-5}$) relative energy spread. All of this is extremely important for the nuclear physics program at Jefferson Lab to advance human understanding of the atom's nucleus.

Visual beam quality control at Jefferson Lab is done with the use of two different types of beam viewers: "classic" and "direct light" beam viewers.

"Classic" beam viewers require thin luminescence screens (for conventional viewers) or carbon foils (for optical transition radiation monitors or OTR) to be inserted in the beam. As a result, the "classic" viewers are invasive and can only be used at relatively small beam currents.

"Direct light" beam viewers at Jefferson Lab are synchrotron light monitors (SLM) and synchrotron light interferometers (SLI). They are based on the use of synchrotron light generated by relativistic electrons moving in the magnetic fields of dipole magnets. The "direct light" beam viewers are absolutely not destructive (not invasive) for the beam.

The resultant beam images from the viewers are captured by TV cameras and displayed in the accelerator control room on the main control display wall as well as on numerous TV screens all over Jefferson Lab. The TV signals are also fed into pipelined high performance image processing systems Maxvideo 200 [1]. The main advantage of the pipeline technology is that the pixel manipulation can be done while the image is being digitized and directed to the image memory. As a result basic image processing operations can be implemented at the full 30 Hz frame rate of the standard NTSC video signal.

MAXVIDEO APPLICATIONS IN ACCELERATOR CONTROLS

All beam image analysis applications at Jefferson Lab are based on the information provided by Maxvideo systems. The systems run a large amount of control and image processing software that has been continuously updated to meet the needs of accelerator operations. The software routinely performs such important functions as masking the pixels outside of the region of interest, subtracting a background image, estimating the transverse RMS beam size and many others. Beam characteristics together with the digitized beam images are entered into a control system database that makes them available for any application running on the accelerator control computer network.

A large variety of high level applications (these applications usually run on workstations) and scripts created at Jefferson Lab, based on Maxvideo systems, allow the users, for example,

- to have a live beam image from any viewer on a monitor of any computer or X terminal connected to the accelerator control network (we call this X-windows application a "beam movie");
- to make a snapshot of any beam image, publish it in an electronic logbook [2], and save it in a file for future reference;
- to calculate the beam energy spread on the basis of the SLI interference pattern and data model as well as to estimate the calculation errors and the data model reliability [3].

Beam image analysis applications significantly simplify beam tuning and diagnostics tasks for the accelerator operations specialists and contribute to very high beam availability for nuclear physics experiments at Jefferson Lab.

At the same time, adding to these applications some functions, which are not typical for "classic" accelerator control software, can significantly increase this contribution.

For example, consider CEBAF accelerator setup procedures. For each machine operational mode, these procedures are very complicated and require extremely careful checking of beam spot sizes and positions on multiple beam viewers. It would be very helpful to have a graphical application that allows the accelerator operations crew to draw simple geometric objects right on top of a live viewer beam image. Such a drawing can show, for instance, the area within which the beam must always be (or not be) on the viewer and become a beam steering template. Once created and saved in a file, such a template can later be used as a reference picture for steering the beam to (or out of) the specified area on the viewer.

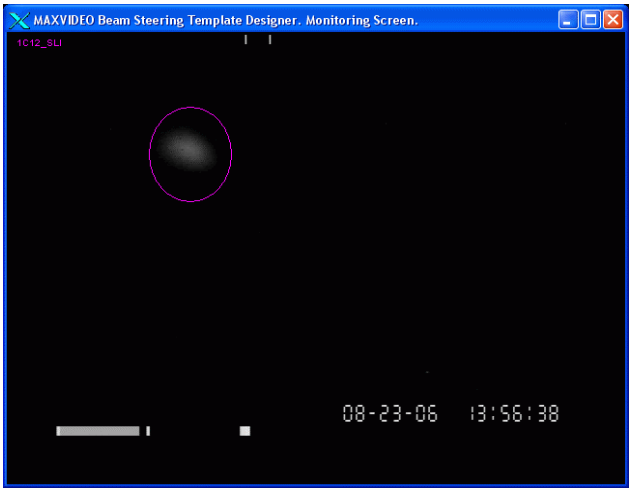


Figure 1: Monitoring screen of the Beam Steering Template Designer application.

If we look for commercial and non-commercial painting software packages available in the market then we find that some of them allow for drawing graphical objects directly over the frames in video sequences. The effects of the use of these packages are amazing. For example, the CinePaint tool [4] was used on The Last Samurai feature film to add hundreds of flying arrows to a battle scene. The main problem is to integrate any of these packages in the control system environment. It is evident that this problem is much more complicated than creating a relatively simple drawing tool as a part of the existing accelerator control and video image processing systems. A prototype of such a tool was created at Jefferson Lab in few weeks. We call it a Beam Steering Assistant Tool.

BEAM STEERING TEMPLATE DESIGNER APPLICATION

The main purpose of the Beam Steering Template Designer application is to create and save beam steering templates for each beam viewer and each machine operational mode. The application provides a user with two information windows: a monitoring screen and a control panel.

The monitoring screen (Fig. 1) has an interactive graphical object window, which is an overlay on top of a digitized live beam viewer image. With the use of a computer mouse, one can easily create and destroy simple color graphical objects in this window. The default object color is magenta. It can be changed when the application starts. The upper left corner of the monitoring screen shows the name of the beam viewer that is currently being used in accelerator operations.

Object manipulations in the interactive graphical object window are handled by the control panel (Fig.2). The panel has information about all functions implemented in the Beam Steering Template Designer application.

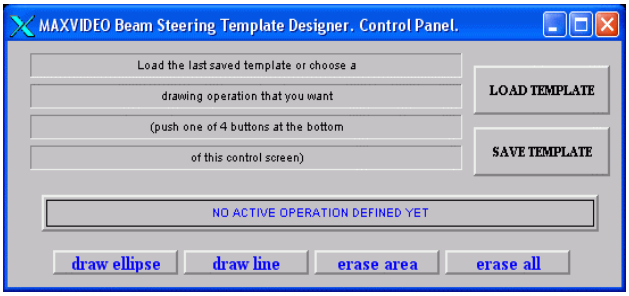


Figure 2: Control panel of the Beam Steering Template Designer application at its startup.

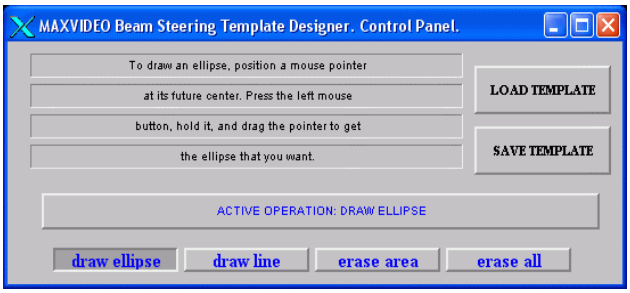


Figure 3: Control panel at work.

When the Beam Steering Template Designer application starts, its control panel looks exactly like one represented in Fig. 2. The bottom part of the panel shows that at this point one can draw ellipses and lines as well as erase created graphics in any area of the window or in the entire window at once.

One can also save graphical objects in a file and load previously created graphics or a beam steering template from a previously saved file. The central status line shows the operation that is currently active. The upper left part of the panel informs the user about what can be done for each active operation or how to activate graphics.

The Beam Steering Template Designer is very easy to use. For example, to draw an ellipse above a live image, you have to press the "draw ellipse" button on the control panel. The panel immediately responds to this command providing the information about what you are going to do

("ACTIVE OPERATION: DRAW ELLIPSE" in this case) and how to do it (see Fig. 3 for details).

We note that, for example, each ellipse is drawn by positioning a mouse pointer at its future center and then dragging the pointer to get the required shape and size. This allows for easy creation of graphical objects with the reference to the beam spot locations on a viewer as it can be seen in Fig. 1.

The software that handles the information windows of the Beam Steering Template Designer is written in C++. It consists of two threads running simultaneously: a main thread and an information thread.

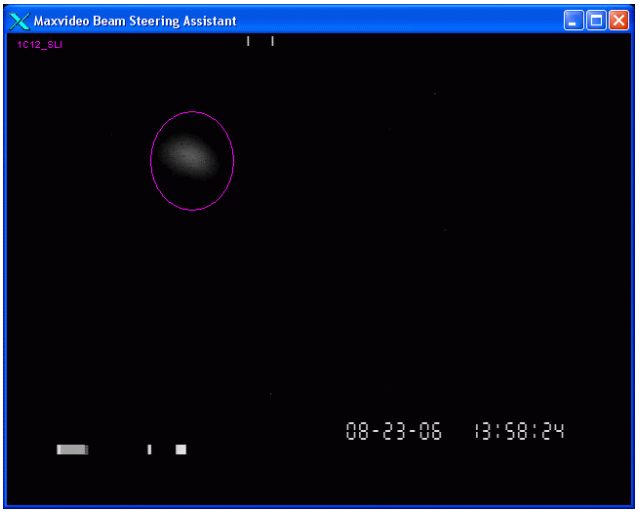


Figure 4: Main information screen of the Beam Steering Assistant application.

The information thread continuously updates the beam viewer images (provided by the Maxvideo systems and control system database) in the graphical object window. It also takes care of the name of the currently used (active) beam viewer, which is shown in this window. Information about active viewers and beam viewer images is a part of the control system database at Jefferson Lab.

The main thread of the software implements the user interface. It handles all the functions of creating and destroying graphics in the graphical object window as well as saving created graphical objects as a beam steering template in a file and loading previously created templates in the window.

Once created and saved, each template can later be used to reproduce the accelerator mode for which this template was created by steering the beam to the specified area on the viewer. We note that for this job the graphics functions of the Beam Steering Designer are not used at all. That is why a light version of this application was created. We call it a Beam Steering Assistant application.

BEAM STEERING ASSISTANT APPLICATION

The Beam Steering Assistant application has only one information window, shown in Fig. 4. During the startup the application requests the information about the beam steering template for the active viewer to use (the default is the last created) and the color to draw this template (the default is magenta) on top of a video image.

The software handling the information window is written in C++ and consists of two threads: a main or viewer image thread and a viewer name thread.

The main thread continuously refreshes beam viewer images in the information window on the basis of the Maxvideo data.

The viewer name thread takes care of the name of the active accelerator beam viewer, which is shown in the upper left corner of the information window. When the active viewer changes, the application updates this name and loads the last created template for this viewer.

In the latest version of software, before loading a template the application pops up an additional small window containing the list of all templates available for this viewer and asks users to choose one. This makes it much easier for users to switch to any desired accelerator mode on the fly.

CONCLUSION

A prototype of a Beam Steering Assistant tool created at Jefferson Lab is an example of implementing some "non-typical" functions in control and data processing systems. The tool, which looks like a simple computer entertaining program allowing users to draw various graphical objects over live TV images, can have a positive impact on accelerator operations. In particular, it can help simplify accelerator setup procedures and make them reproducible.

REFERENCES

[1] P. Chevtsov, "Multivideo Source System for Beam Diagnostic Applications" PCaPAC-2000. DESY. Hamburg. Germany. 2000.

[2] T.Larrieu, T. Mcguckin, "Beyond an Electronic Logbook". PCaPAC-2005. Hayama. Japan. 2005.

[3] P. Chevtsov, "Automated Image Quality Optimization for Synchrotron Light Interferometers", ICALEPCS 2005, Geneva, Switzerland, 2005.

[4] www.cinepaint.org

[5] www.trolltech.com

APPLICATIONS OF INTEREST: A RELATIONAL DATABASE APPROACH TO MANAGING CONTROL SYSTEM SOFTWARE APPLICATIONS*

D. Quock#, N. Arnold, D. Dohan, J. Anderson, D. Clemons, ANL, Argonne, IL 60439, U.S.A.

Abstract

Large accelerator facilities such as the Advanced Photon Source (APS) typically are operated by a diverse set of integrated control systems, such as front-end controllers, PLCs, and FPGAs. This type of control system structure encompasses numerous engineering documents, distributed real-time control system databases, source code, user displays, and other components. The complexity of the control system is further increased as the life cycle of a control system is never ending, change is constant. And the accelerator itself generates new operational problems on a regular basis. This overall controls environment begs the question of how best to provide a means for control system engineers to easily and quickly troubleshoot unique functions of the control system, find relevant information, and understand the impact of changes to one part of the control system on other applications. The answer to this question lies in being able to associate pertinent drawings, manuals, source code, hardware, and expert developers in an efficient and logical manner. Applications of Interest is a relational database software tool created for the purpose of providing alternative views of the supporting information behind each distinct control system application at the APS.

IRMIS

The foundation for the Applications of Interest (AOI) project is the collaborative effort between several Experimental Physics and Industrial Control System (EPICS) sites to build a common relational database (RDB) schema for documenting large and complex particle accelerator control systems. The result of this collaborative effort is the Web-based relational database software application Integrated Relational Model of Installed Systems (IRMIS) [1]. At the Advanced Photon Source, IRMIS is a collection of Java, PHP, and Perl tools that search and populate a MySQL relational database. The IRMIS database stores information about programmable control devices such as EPICS IOCs and programmable logic controllers (PLCs), and the process variables and interconnecting hardware shared among these devices. To enable intuitive and quick access to control systems information, the user interface for IRMIS is organized into the separate viewers: IOC, PLC, Component Type, Network, Controls Spares, PV Information, Installed Components, Cables, and Applications of Interest. Access to the IRMIS viewers is from the IRMIS main display as shown in Figure 1. Interactive links from one IRMIS viewer to another are provided where appropriate.

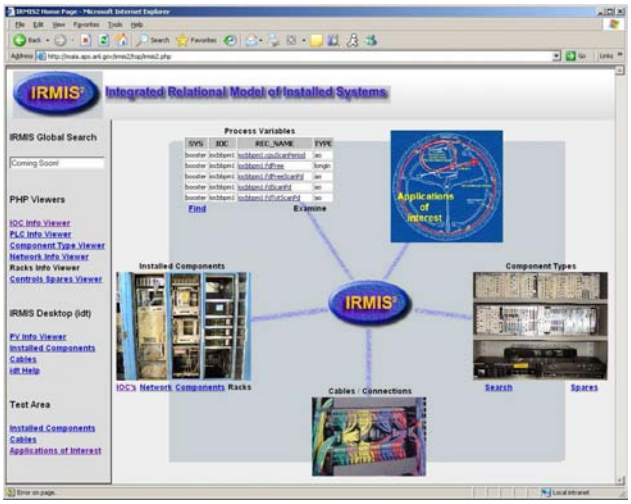


Figure 1: IRMIS Main Display.

APPLICATIONS OF INTEREST

The motivation for creating an Application of Interest view of the controls system at APS was to provide a means for establishing a starting point for control systems developers to navigate through the various pieces of distinct controls systems applications. Examples of distinct control system applications viewed in this manner at APS include the storage ring vacuum control system, storage ring vacuum bakeout system, linac water leak detection system, and linac timing system. Over 130 AOI control systems have been cataloged thus far, and it is expected that this number will exceed 300 when the controls system for the entire APS site is fully documented.

AOI Attributes

To characterize each Application of Interest control system, several basic attributes are defined by control system developers and stored in the IRMIS relational database (see Table 1). These attributes are entered through an AOI Editor Web-based display built in PHP (see Figure 2). Other attributes of an AOI are automatically discovered or generated by an AOI Crawler and include EPICS process variables, user-programmable components (UPCs), and EPICS IOC startup command lines that support the particular application.

*Work supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under Contract No. DE-AC02-06CH11357.

#quock@aps.anl.gov

Table 1: AOI Attributes Defined by Controls Developer

Attribute	Example
AOI Name	aoi_sr_mps_system_latch-card
Cognizant 1	Marty Smith
Cognizant 2	Ned Arnold
Customer Group	Controls
Criticality	2 Equipment or Beam Inhibit Risk
Description	The latch cards are installed in 20 VP IOCs around the ring.
Status	Active
Functional Criteria	A validation procedure confirms the functionality of the system after every shutdown.
Keywords	MPS, latch card, beam inhibit
Notes	Request was made to Operations Group on 10/29/06 to add list of process variables to dump data printout following an MPS trip.
EPICS Top Displays	- /usr/local/iocapps/adlsys/srbpm/miscApp/mainMPS.adl - /usr/local/iocapps/adlsys/sr/mpsApp/MPS-latchMasterPanel.adl
Revision History	Storage ring MPS latch card consolidation project began April/May 2006 shutdown. New EPICS sequence program was installed for sectors 1 through 14, and 35 through 40.
Documents	ICMS

The AOI attribute AOI Name is the main crux behind linking the AOI Viewer to other APS documentation software applications such as the IRMIS viewers, the APS Integrated Content Management System (ICMS), and the Controls Logbook. By implementing a strict naming convention for AOI names, the AOI name can be used to quickly locate pertinent engineering documents (located in the ICMS repository), search reported troubleshooting measures (located in the Controls Logbook), and trace closely associated AOIs (located within IRMIS).

A two-level hierarchy relationship of parent-child is used to identify closely associated AOIs. This follows from the object-oriented design concept of inheritance. Here, a child AOI is a specific instance of a parent AOI, but the children of a given parent may have a wide variety of differences that make them distinct from each other. An example of this at the APS is the parent AOI:

aoi_linac_diagnostics_flag

and its children AOIs:

aoi_linac_diagnostics_flag_fs-1
aoi_linac_diagnostics_flag_fs-4
aoi_linac_diagnostics_flag_fs-7
aoi_linac_diagnostics_flag_fs-9

In this example, fs-# refers to a unique flag station number located in the linac.

The benefits of having a parent-child relationship for AOIs (when suitable) is to aid in searching for the desired amount of detail of a specific control system application. A parent AOI will provide information on broadly defined EPICS displays, engineering specifications, revision history, etc. In other words, the view provided by the parent AOI gives an engineer a quick and general overall understanding of the controls system at hand. The child AOI provides information on and links to more detailed information specific to that installation and operational issues.

The AOI name is derived from a carefully defined set of fields as follows:

aoi_<machine>_<technical system>_<unique function>_<child>

The fields "machine" and "technical system" are defined in tables in the IRMIS database and are specific to the APS site. The "unique function" and "child" fields are free text entry defined at the discretion of a controls developer's expertise of a particular control system. The "child" field is optional and depends on the complexity and inheritance nature of a controls system application as explained above. Each AOI name is forced to be distinct by the AOI Editor and is a requirement built in the IRMIS database.

The AOI attributes Criticality and Status are predefined lists located in the IRMIS database. They are included in characterizing an AOI in an effort to aid management when identifying control systems that are of higher operational priority for assigning engineering staff, enforcing thorough documentation, and tracking development and upgrades of AOIs. The levels for Criticality range from 1 to 5 with 1 being the most severe. The AOI Criticality choices are:

1. Personnel Risk [e.g., Radiation Safety System]
2. Equipment or Beam Inhibit Risk
3. Beam Performance Risk
4. General Operations
5. R&D [e.g., Test Stand]

The options for AOI Status are: Active, Inactive, Decommissioned, Under Development, and Other. The remaining AOI attributes are self explanatory and are shown with examples in Table 1.

AOI AT WORK

To better appreciate the usefulness of the Applications of Interest IRMIS viewer, a real-life accelerator operations troubleshooting example will be walked through here. Consider the situation where an operator receives an alarm on a Machine Protection System (MPS) fault on a weekend, and the fault occurs only once and is not reoccurring. Given that there are 800 input signals to the storage ring MPS latch cards at APS, it can be a daunting task for the operator to determine easily if an

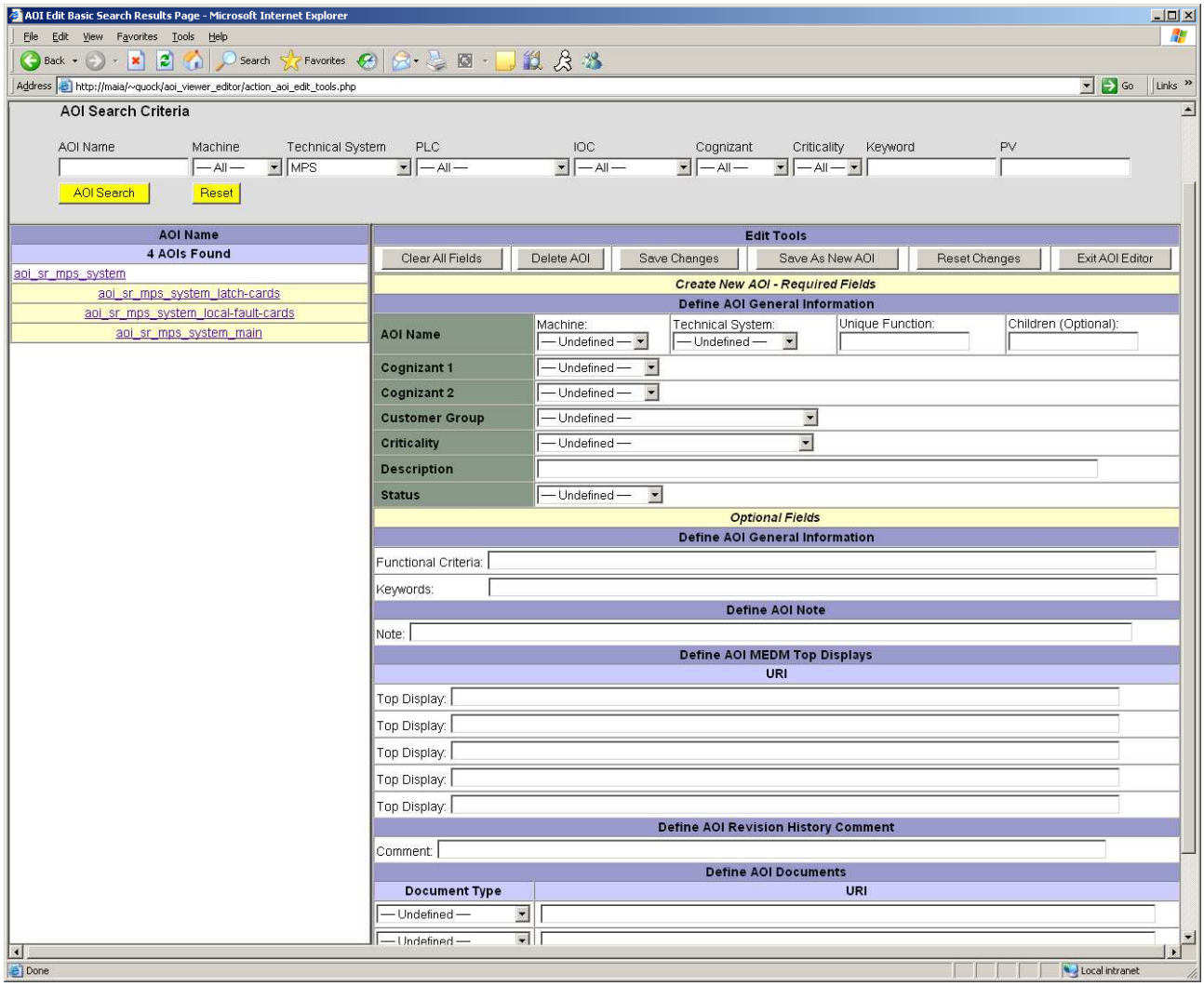


Figure 2: AOI Viewer.

operating set point was actually exceeded, or if one of the latch cards or its associated cabling caused an erroneous trip of the MPS. The operator notifies the controls system engineer on call for the weekend, and thus passes on the task of identifying the underlying cause of the MPS fault, and the possibility of further disruption to the beam delivered to users. And it just happens that the engineer on call this weekend is not the MPS expert.

The engineer uses the best tool available for rapid controls system troubleshooting -- Applications of Interest. Through her Web browser, she launches the AOI Viewer and decides to use the AOI Search selection bar for efficiently narrowing down available documentation resources pertinent to the situation. As shown in Figure 2 above, by selecting MPS from the drop-down Technical System category of an AOI as the sole search criteria of the IRMIS database, four AOIs are returned as a result set. Of the four AOIs, all have the technical system MPS included in their name. One of the children AOIs has latch-cards as the child name field. Information on this specific AOI is then just one click away. By selecting the child AOI aoi_sr_mps_system_latch-cards (as shown on the left side of Figure 2), all of the attributes for this specific AOI are displayed (see Table 1) including recent information about changes made to this control system application, which EPICS displays to use for viewing real-time readbacks of MPS latch cards, and which IOCs are associated with each storage ring sector and, thus, each latch card. Other IRMIS viewers can then be used to trace cabling and other hardware associated with an IOC.

In conclusion, the IRMIS viewer Applications of Interest enables intuitive and quick access to control system information that is needed for troubleshooting operation issues, training engineers, managing controls staffing requirements, and tracking development and upgrades to control system applications.

REFERENCES

[1] C. Saunders, D.A. Dohan, and N.D. Arnold, “The IRMIS Object Model and Services API,” ICALEPCS’05, Geneva, Switzerland, October 2005.

[2] D.A. Dohan, N.D. Arnold, “IRMIS,” http://www.aps.anl.gov/epics/irmis.

ACCELERATOR MANAGEMENT WITH WEB-BASED GIS

A.Yamashita, Y.Ishizawa, T.Ohata, M.Takeuchi, SPring-8, Sayo, Hyogo, Japan

Abstract

We are developing accelerator management systems based on Geographic Information System (GIS). An open source GIS system, MapServer, integrates almost every location related information around SPring-8 accelerator complex. Users enjoy its Google-map like functions, zooming and panning, in their web browsers. MapServer accepts various kind of data format and data sources. It reads vector and raster, JPEG to CAD data, filesystem and relational database management system. Not only static data like equipment location, the system also handles dynamic real-time status of accelerator components. We build two systems. One is SPring-8 equipment management system and another is SCSS (SPring-8 Compact SASE source) alarm system We will discuss the development of those systems.

INTRODUCTION

The google map [1] impressed us by its smooth map handling. It implements zoom and pan on web browsers using Ajax technique. It also displays informations linked to map coordinates. We wish to have *Google map for SPring-8*. Because SPring-8 is a large facility and we have many disorganized data which should be liked to their location.

SPring-8 has been operating since 1997. The maintenance history such as repairing and replacing equipment have been recorded by individual persons. Those informations are stored in Excel files or text files in distributed individual personal computers. Those disorganized data are sources of confusion in every-days operation.

The pioneering research of T.Larrieu et al. presented at ICALEPCS2005 [2] inspires us to use GIS (Geographic Information system) to organize those equipment informations linking with their location in SPring-8. We found very few other projects which uses GIS for small area, like accelerator facilities. [9]

GIS

GIS has wide variety of definition. The Wikipedia [4] defined GIS as follows.

> A geographic information system (GIS) is a system for creating, storing, analyzing and managing spatial data and associated attributes. In the strictest sense, it is a computer system capable of integrating, storing, editing, analyzing, sharing, and displaying geographically-referenced information. In a more generic sense, GIS is a tool that allows users to create interactive queries (user created searches), analyze the spatial information, and edit data.

We began to study to use integrating, storing and displaying functions of GIS.

Requirements

We required following conditions for the GIS system.

- Web based display client and server system.
- Open source.
- Handle CAD data we already have.
- Easy to manage both creating application and data maintenance.

MAPSERVER

We choose MapServer [3] satisfying those requirements. Many commercial products match to above conditions except open source. A few projects in open source. MapServer was originally developed at University of Minnesota for managing forest resources. It integrates various image format, vector and raster and attributes in text format and relational databases. It is becoming very popular in GIS community. Two books have been published on MapServer [6], [7] in United States. One book was translated in Japan [8]. We do not explain details of MapServer functions here. Rich number of informations available from Internet and those books.

MapServer system provides only functions, called MapScript, which receives request parameters such as coordinates, display areas, layers, etc. and generates temporary image file. User can build applications on it. The application is not necessary web application, but also standalone application. MapServer has APIs for many languages like PHP, perl, Python, Java and Tcl/Tk. Many products have been implemented on MapServer API[1].

SYSTEM DEVELOPMENT

We have been developing two systems for SPring-8 equipment manager and SCSS test accelerator [10] alarm display system. The common system architecture is shown in Fig. 1. We build both systems based on p.mapper [11] a system for web browsing client written in DHTML and server written in PHP/MapScript. We chose p.mapper from wide variety of selections because of its rich functions.

[1] For example, http://www.maptools.org/ site has list of MapServer applications.

The MapServer is able to run in multi-platform. We developed systems on a Windows 2000 machines. We checked it was running on Linux server without any modifications. P.mapper client is written in Javascript taking account of incompatibility between Internet Explorer (IE) Mozilla browsers. We tested our systems on IE, Mozilla Firefox and Opera browsers.

1. A user sends requests from a browser application written in DHTML.

2. The http server sends cgi command to the p.mapper server written in PHP.

3. The p.mapper server make API call to the MapServer.

4. A configuration file for MapServer organizes data files and RDB data.

5. MapServer reads image data and attribute data from data files or RDB servers using the mapfile.

6. MapServer generates temporary image file to display in the browser.

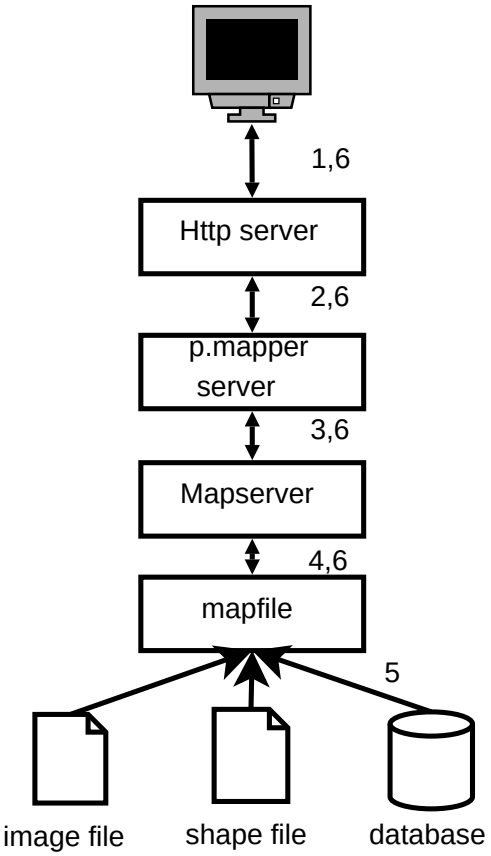


Figure 1: MapServer system

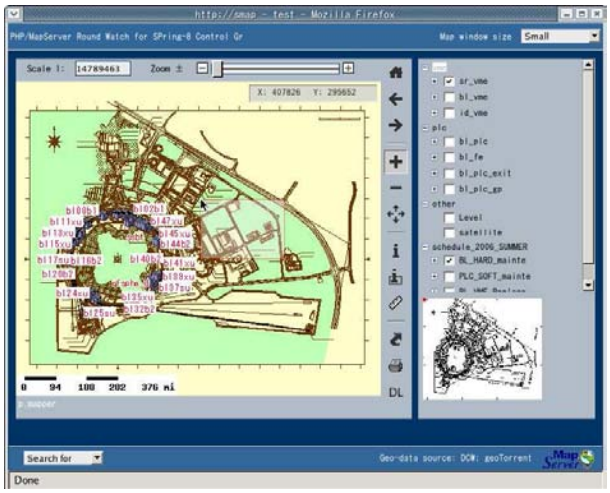


Figure 2: Screenshot of SPring-8 equipment management system.

Data Sources

We converted CAD data files to MapServer readable shape files format both systems, because MapServer cannot handle CAD data directly. We used fGIS [5], a free application, to convert AutoCAD .dxf file formats to shape file format. At first, we removed unwanted data from original CAD data by hand and converted to shape file. fGIS exports separated shape files for polygon , line, multi-line and point layers. We use polygon and line layers. We stored a part of polygon data into RDB to link to other data. Most of layer data are saved as shape file format because of the access speed. Mapfile integrates them to generate map images.

SPring-8 equipment manager

We build SPring-8 equipment manager to manage equipment distribute in SPring-8 site. Currently, the system manages VME system and PLCs'. The system manages following items.

- VME and PLC location
- Slot management of VME.
- Maintenance history and plan of VMS and PLC

A ProtgreSQL relational database server manages those attributes. The system groups VMS and PLC and displays in different layer on the map. User can select layers by buttons displayed on web browsers. P.mapper provides other rich functions like distant measurement, hyper link from icons on the map, zooming using DHTML function, reference map and etc.. A screenshot of the equipment management system is shown in Fig. 2.

SCSS alarm display system

We use GIS system for real-time and dynamic display of the data. SCSS successfully observed the first lasing in

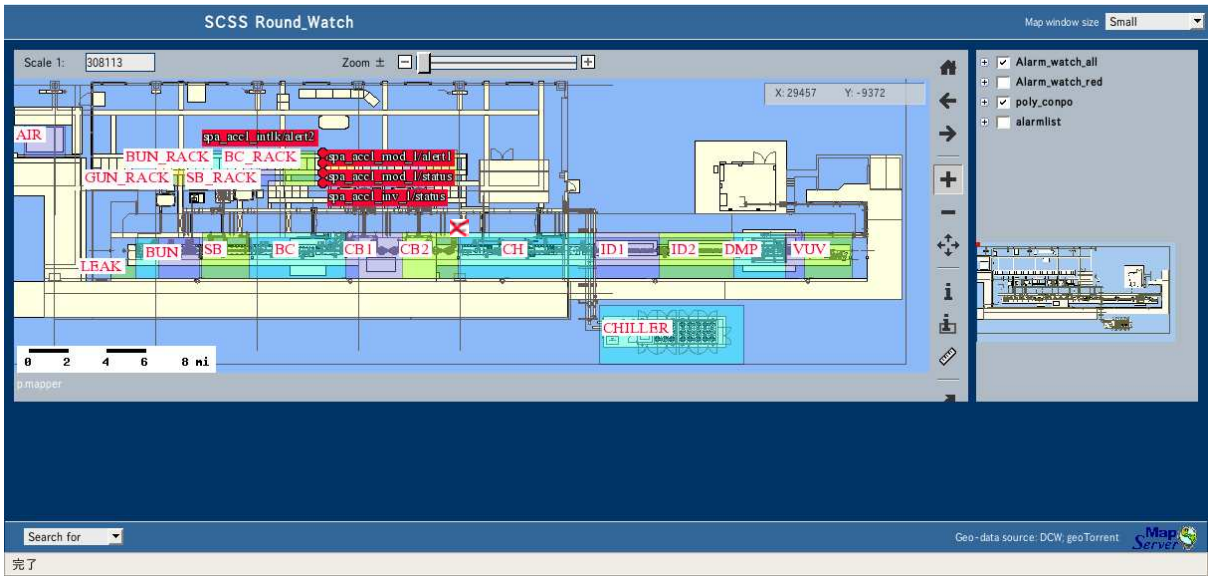


Figure 3: Screenshot of SCSS alarm system

June 2006. We applied standard SPring-8 alarm system[12] to monitor SCSS equipments. It monitors data which were acquired periodically from SCSS and writes alarm data to the SPring-8 control database when unusual data were discovered. We developed graphical alarm display system to display the name and location of alarmed signal on the interactive map. A database client process periodically reads alarm data from the database and writes alarm file for MapServer. The alarm file contains name, location and alarm level of alarmed signal in the GML (Geography Markup Language) file format[13]. The client program running on the browser periodically reloads map image from the http server without reloading entire web pages using Ajax techniques. This system also employs shape file converted from CAD .dxf file as base data. Alarm system monitors 686 signals of SCSS. Because we have no time to enter location of each signals to database. We divided map into 26 areas and make signals belongs those area. The alarmed signals are displayed in the area on the map. Screenshot of the SCSS alarm system is shown in Fig. 3.

Development

After we decide to develop a system using MapServer, it took about one man month to build the equipment manager. Most of the time was spent to understand mapfile. In addition, at the beginning we have very little knowledge on PHP and Javascript. One month includes learning those languages. On the other hand, the development of SCSS alarm system was straightforward. We developed the system only one week. After we learned how to use MapServer the development became quite easy. However, entering data points takes much man power. After entering VME and PLC data, data entry for other equipment is under way.

CONCLUSION AND FUTURE PLAN

We developed two google map like web based map system for our facility. Open source products MapServer provides easy way to develop and integrate data distributed around into one map system.

We are now developing easy data entry system for everybody enters location and upload their text data, photo, drawings from their browser. It reduces data entry tasks which is considered to the biggest problem in GIS.

REFERENCES

[1] http://maps.google.com

[2] T. Larrieu et al., “Evaluating the Potential of Commercial GIS for Accelerator Configuration Management“, ICALEPCS 2005, Oct 2005, Geneva.

[3] http://mapserver.gis.umn.edu/

[4] http://en.wikipedia.org/wiki/Gis

[5] T. Mitchell, “Web Mapping Illustrated“, Oreilly, Jun 2005, Sabastopol.

[6] B. Kropla “Beginning MapServer“, Apress, Aug 2005, Berkeley.

[7] T. Mitchell, K.Otsuka (Translation) “Nyuumonn Web Mapping“, Oreilly Japan, May 2006, Tokyo.

[8] M. Matias et al., “Evaluating MicroStation GeoGraphics GIS“, IWAA 2006, Sep. 2006, Menlo Park.

[9] T.Shintake, “Status of the SCSS Test Accelerator and XFEL Project in Japan“, EPAC’06, Jun 2006, Edinburgh

[10] http://www.pmapper.net

[11] http://www.forestpal.com/fgis.html

[12] A.Yamashita et al., “The Alarm System for the SPring-8 Storage Ring“, ICALEPCS 1997, Oct 1997, Beijing

[13] http://www.opengeospatial.org/standards/gml

STATUS OF THE CEBAF CONTROL SYSTEM*

M. Bickley, Jefferson Lab, Newport News, VA 23606 USA.

Abstract

The Continuous Electron Beam Accelerator (CEBA) has been in operation at Jefferson Laboratory since 1994. The evolution of the control system since that time has resulted in a steady increase in the scope of control being executed on commodity computer hardware, mostly during the last six years. We have installed a number of single-purpose Linux-based computers that provide infrastructure capabilities such as name services, gateways and archiving. The large-format display, central to the machine control room, operates on a high-end personal computer. In the real-time arena the lab is pursuing inexpensive PC-104 daughter boards to serve as control system interfaces to an FPGA-based fast data acquisition system. Our expectation is that the digital RF system required for the lab's planned 12 GeV upgrade will have a similar design. This paper will discuss these and similar solutions, highlighting how they enable reliable operation of the accelerator.

THE ORIGINAL CEBAF CONTROL SYSTEM

At the start of CEBAF commissioning the control system used at Jefferson Laboratory was, in many ways, the antithesis of modern, open control systems. Instead of the now standard three tier construction we implemented a two-tier architecture. We had front-end Hewlett-Packard (HP) computers controlling hardware through an HPIB interface connected to CAMAC data acquisition crates. On the back end we used Hewlett-Packard computers serving as operator displays.

The control system software implementation choices we made then were also not in line with current thinking. We implemented a completely closed system, built of single-vendor proprietary components. The data acquisition and control software depended on HP-specific shared memory interface software. The communication protocol between the front-end and back-end systems utilized HP-specific networking capabilities. On the back-end systems the same shared-memory software was used, and the operator interfaces all depended on HP-proprietary graphics libraries.

In sum, our control system was completely dependent on both HP hardware and HP software. This greatly limited the flexibility of the control system, put the accelerator controls at the mercy of Hewlett-Packard's developmental plans, and isolated us from advances being made throughout the accelerator controls community. The past 12 years have seen the laboratory make great progress towards an open, modern control system. The remainder of this paper will point out some of the steps we have taken towards increased standardization and openness, and discuss future plans that continue in that direction.

FRONT-END SYSTEMS

Jefferson Laboratory has made great progress from the time that all control system front-ends were HP computers running custom-built data acquisition software. Now we use almost entirely VME-based single-board computers. These computers run the VxWorks operating system, with EPICS providing data acquisition and control. The use of EPICS in the control system provides the lab a more open environment, thanks to changes made within the EPICS collaboration.

The last few years have seen the EPICS developers create an operating system independent (OSI) layer in the real-time control portion of the software. The result of this change is that EPICS' front-end dependence on VxWorks has been severed. Now an input-output controller (IOC) can execute on any of a variety of operating systems, including VxWorks (of course), Windows, Solaris, Linux and RTEMS. Porting the IOC code to a new operating system requires only that the OSI interface be developed for the new system. The more posix-compliant the new operating system is, the more straightforward it is to implement the port.

PC Hardware and Low-level Controls

We have leveraged the EPICS changes to take additional steps towards opening the CEBAF control system. We now use a handful of PC104 IOCs, small-format standard PC cards such as shown in figure 1, running EPICS on top of a standard linux operating system (Red Hat Linux). Due to the poor determinism and lack of prioritization flexibility in this version of linux, this implementation is only useful in situations where real-time response is not required. At Jefferson Laboratory we use front-ends like this to control slow devices, such as a thermal mass-based calorimeter.

Figure 1: A typical PC104 card

The lab is pursuing much more widespread use of PC104. Because this hardware implements the ISA standard and uses standard Intel processors, these systems are much less expensive than comparable VME-based solutions. The RF group at the lab wants to have a general-purpose microprocessor dedicated to the execution of all of the front-end controls for each of the 384 cavities used for electron acceleration. This would be prohibitively expensive using VME, but is much more reasonably priced when based on commodity hardware. While VME-based solutions are a good fit for high channel-density applications, low-density applications are an excellent fit for commodity hardware like PC104.

Jefferson Laboratory is investigating this solution using a similar hardware configuration dedicated to the control of a fast data acquisition board used to monitor RF signals at rates from 1 to 10 megahertz. This implementation uses RTEMS rather than linux as its base operating system. Because RTEMS is a true real-time operating system, we can develop much more deterministic controls and higher data rates than would be possible with linux as the underlying operating system.

An effective implementation of the fast data acquisition application has positive implications for the long-term future of the rest of the CEBAF control system. The open-source operating system RTEMS has board support available for all of the different real-time control hardware used at the lab. Consolidating all of our real-time controls under a single, non-proprietary operating system will enable us to simplify the control system support environment. At the same time we will be better able to match the control hardware used against the data acquisition and control requirements of the specific application.

THE MIDDLE TIER

The middle tier of the CEBAF control system is much more flexible than either the front-end or back-end tiers, as is the case in most control systems. The software executed in the middle tier is limited in scope because there are no user logins. The interfaces they present to the rest of the control system are well-defined, so replacing a component of the middle tier requires only that it support the same network interfaces.

Until 2000 the middle tier of the accelerator control system was implemented entirely on HP computers. At that time the lab needed to upgrade the systems hosting the middle tier. It was clear that the most cost-effective computing solutions could be found on commodity PC hardware. Those systems had compute power and memory capacity at least equivalent to the HPs then in use, and were available at a fraction of the cost. On top of those savings, the annual licensing fees were reduced dramatically as well through the use of the linux operating system.

The Jefferson Laboratory Archiver

The first step in migrating away from Hewlett Packard computers involved the Jefferson Laboratory archiver, *czar*. This service was chosen for two reasons. First, it was simple to operate multiple, duplicate archivers, one HP and one commodity system, in parallel. This made it straightforward to execute performance comparisons on real data. The second reason we chose the archiver was that we projected significant increases in demands on the system. A sufficiently-powerful HP computer would have been more expensive than a commodity computer by an order of magnitude; this would have been prohibitively expensive.

Our experience was overwhelmingly positive. We were able to operate dual archivers for a prolonged period. When we were fully confident in the reliability of the new hardware and the performance gains it presented, we were able to transition without any impact on our user base.

Additional Services

Given the positive results from our first effort, it was clear that this was the right long-term direction for the laboratory. Over the following seven years the remainder of the accelerator's middle-tier services were migrated off of Hewlett Packard computers. These included control system-specific services such as model servers, orbit locks, energy locks and database servers. Typically the transition for these services was somewhat more difficult than it had been for the archiver. Since only one of each service can be operational at a time, testing and transition time had to be integrated into the accelerator running schedule.

BACK-END SYSTEMS

The CEBAF control system back-end computers continue to be nearly all Hewlett-Packard, running X-windows. Because the accelerator has been in operation for more than 10 years, proliferation of user tools makes migration away from HP more problematic than the middle tier systems. A large set of software and files, including not just binary executables but also scripts, libraries and binary data files, have to be managed in an operating system-independent fashion prior to the start of a smooth migration. That process has been set in motion, with roughly 80% of the total effort complete.

The exceptional back-end system is used to power the large-format display wall shown in figure 2. The display is an 8-cube array of DLP-based monitors, with a resolution of 4096x1024 pixels. To the controlling computer the array looks like a very large, single display. The system is a standard Intel-based personal computer with special graphics management hardware to provide its unique capabilities. Graphics cards can be added to the system to enable standard video outputs to be sent to windows on the display. Each card supports four video input channels, and four cards can be added to the system, providing a total of 16 video windows integrated into the standard display of the wall.

Figure 2: The CEBAF Control Room Display Wall

The display wall and control hardware enables accelerator operators to exercise any of their standard control tools and easily monitor the response on standard video outputs, like those generated from viewers or synchrotron light monitors. Future capabilities will include the integration of remote camera outputs so that operations staff can monitor the state of hardware and the environment from the control room. This will enable them to view, for example, conditions in the accelerator tunnel, looking for leaks or unsafe conditions, without requiring an access and radiation survey. The result will be increased time dedicated to operating the accelerator, and meeting the needs of the accelerator's customers.

CONCLUSION

The last seven years have seen a significant increase in the use of commodity hardware and software throughout the CEBAF control system, from front-end data acquisition and control computers to back-end monitoring and display systems. These efforts have been consistently successful. Looking ahead, we see many more opportunities to take advantage of the inexpensive power and flexibility that are possible with open architecture hardware and software. Taking these steps will not require making any concessions in functionality or performance. Our experience to date has been that we can leverage the work of other laboratories to obtain solutions that meet all of our users' requirements in a highly cost-effective manner.

CONTROL SYSTEM STUDIO (CSS)[1]

M. Clausen, Jan Hatje, Markus Möller, DESY, Hamburg, Germany

Abstract

The current applications for operators working with the EPICS toolkit have been designed, when UNIX workstations were state of the art and X-Window provided the preferred libraries to create graphical user interfaces. The next generation of OPI's should be designed to run on any operating system and for most control systems. The look and feel of all applications should be the same and it should be possible for the operator to gain access to any relevant information across the installed applications seamlessly. The Control System Studio (CSS) is under development to fulfill these requirements. It provides core functionalities like management services (authentication, authorization, application updates ...) and central logging services. A common Data Access Layer (DAL) provides an interface for transparent access to any control system data interfaced to the DAL. Basic CSS design works and the DAL implementation have been subcontracted for the most professional implementation. CSS is implemented as a set of plug-ins in the Eclipse Rich Client Platform (RCP) and thus written in Java.

MOTIVATION

Today's operator interfaces mostly consist of a set of independent tools or programs. These kinds of programs can only be executed on specific operating systems (OS) like Windows or UNIX. They are tightly connected to the control system they are written for. Programs written for a specific platform can still vary a lot in terms of look and feel or ease of use. The motivation for a completely new design of a Control System Studio is driven by these factors:

- Operating System independent
- Common look and feel
- Decoupling from control system specific interfaces

CSS DESIGN

An operating system independent implementation of programs can be easily achieved by using Java as the programming language. The selection of the development environment and the implementation framework were even harder decisions to make. Two market leaders share the two regimes of Integrated Development Environments (IDE) and runtime frameworks: Eclipse and Netbeans.

Eclipse as an IDE and RCP Framework

A thorough investigation was carried out between the two candidates. As a result the leading IDE – Eclipse – was chosen not only for programming but also as the rich client platform (RCP) framework. Having made the decision in 2005, the actual performance of Eclipse developments and the rapidly growing community of Eclipse programmers has shown that this was a good choice. Last not least it also adds momentum to the ongoing CSS-developments in the control system community.

Common Look and Feel

Common look and feel is one of the basic design features of the CSS. Eclipse already provides an environment which eases the usage of sharable applications. But it still leaves a lot of freedom to the implementer how internal functionalities are implemented and activated. This finally lead to the decision that CSS applications should not be implemented as individual Eclipse applications, but the applications should be implemented as Eclipse plug-ins and be activated from within the CSS. This approach provides a high level of integration (see: Information 'On Your Fingertip'; Drag and Drop Data across Applications), but it still leaves the freedom to activate applications (plug-ins) outside the CSS framework – at a reduced integration level.

Information 'On Your Fingertip'

There's a saying that you do not have to know everything – you just have to know where it's written. The same applies if you need to get access to detailed information about control system components. The 'Information on your fingertip' – or 'Object Aspect' - can be accessed by pressing the right mouse button (MB3). A menu will pop up which shows all the applications which can be called up to get more information about the component you just click on. This way the user (operator) can call up applications which provide access to information stored in configuration files, LDAP servers, relational databases or web pages. Other types of applications comprise debugging- or configuration tools. This functionality is implemented using an Eclipse core functionality known as 'contributions'.

Drag and Drop Data across Applications

Besides the MB3 control, another basic CSS functionality will ease the use of multiple applications within the CSS framework. Transferring data between individual applications will enable a seamless integration of applications. This functionality will allow dragging data of a specific type into another application which has implemented the complementary interface to accept this type of data. This way it is only possible to drop data into applications which are prepared to process them. The data types span from simple device names up to complex data structures like arrays of archived data.

Managing CSS Installations

Managing individually installed applications on a wide spread set of PC's and/or Thin Client machines can be a nightmare if the tools to service these installations are not

available. Eclipse comes with a built in automatic update facility based on Web-based update sites. This functionality eases the synchronous update of all CSS instances that have access to that Web site. In a controls environment the requirements for updating individual CSS instances can be quite different. While the update of applications installed in the office area might be no problem, updates in the control room might interfere with current machines operations.

A mechanism will be part of the CSS-core functionality to manage CSS updates and to monitor CSS instances for bug tracking and their usage of system and/or control system resources.

Interfaces - Interfaces

In order to keep CSS independent from specific implementations – or even control systems, one of the basic design goals is to leave out control system specific code from CSS core. This way it will be possible to share applications – as long as they are designed based on common interfaces rather then specific implementations. Even though it's highly desirable to follow this design pattern, it will not inhibit developers to write their control system specific CSS plug-in. Specific interfaces are:

- The Data Access Layer (DAL):
  The DAL provides a homogeneous interface for CSS applications on one side and a control system interface on the other side. The DAL implementation supports channel based as well as device based control systems. Collections of channels can be comprised to devices on the DAL level. The DAL is an CSS independent library.
- Authentication:
  Authentication is one of the CSS-core interfaces. CSS-plug-ins can be written to implement the authentication scheme of the local site. Several implementations for e.g. Kerberos, or even local file access will be part of the basic CSS-core package.
- Authorization:
  Authorization is another CSS-core interface. Example implementations include an LDAP interface as well as a simple configuration file.
- Name service interface:
  Name services and namespace browser will use this interface. Based on JNDI it is easy to integrate services like LDAP or other similar implementations.
- Archive Access Layer (AAL):
  The AAL is a similar implementation like the DAL. Once the corresponding AAL-plug for the specific local data archive has been written, it will be possible to use the CSS archive viewing tools.

Authentication and Authorization

Most control systems provide their own security mechanism. One might ask why another authorization mechanism is foreseen for CSS. Many actions from the operator panel need a very fine access granularity which might not be based on the accessed device but on the expertise of the user. This kind of access control can be implemented on the application level itself. The interfaces mentioned before provide the hooks to add access control to the levels of graphical widgets and commands issued from the graphical user interface down to individual classes.

A secure store for passwords will add another support level to the CSS core. This way the user can authenticate for CSS usage and in addition for access control to other applications or systems which shall be accessed from within the CSS framework.

Collaborative CSS Development

Even though CSS developments have been initiated by DESY, it's desirable to keep CSS as open as possible for collaborative contributions. This criteria is not only an organizational aspect for the CSS design it also influences the software design. For instance the choice to use the authentication interface – including the login window – must be left open to the software implementer – or even the project manager – of the local site. Flexibility is key to keep the acceptance for CSS high.

CSS APPLICATIONS

The flesh and blood of CSS are the applications which are running inside the CSS environment. A lot of smaller test and debug applications have already been created in the current design and development process. But CSS will only come to life when core applications are available.

Synoptic Display

The synoptic display can be clearly identified as the 'killer' application for CSS. Due to the importance of this application, it has been outsourced to the Institute of Informatics of the University of Hamburg. The requirements have been collected in the course of a CSS workshop held August'2006 [2] at DESY. Only a few of the requirements can be highlighted here:

- Graphics can be created by non professional programmers and thus are configured – or graphically programmed as one might call it.
- The persistent store of configuration data is kept in XML files.
- New graphic elements can easily be added to the CSS graphic framework. Ideally new elements have only to implement two extension points for the edit mode and the runtime mode.
- Display call up must be accomplished within the time frame of a second.
- Extended edit features. As a result of a survey of editing frameworks - GEF has been choosen.
- Support for zooming, panning and multiple graphic layers.

Alarm Displays

The CSS Alarm Displays are developed on the basis of a new alarm scheme. An alarm 'push model' will be in

place to catch all alarms from the control system's front end controllers. An alarm system based on the Java Message System (JMS) will pass alarm messages to several different kinds of alarm destinations. One of them being the CSS Alarm displays. The two basic displays are:

- Alarm Tree View. It is set up by reading the configuration of a hierarchical device tree from a JNDI-based store (like LDAP). The Alarm Tree is implemented in a way that the top branch shows the state of the most critical alarm of any of the subsequent branches down to the leaves.
- Alarm Table View. It shows incoming alarms in a table. The alarms are prioritized by the severity and then ordered by the time of their occurrence. This way the operator has always an overview of the most critical alarms. Alarm acknowledgements across several CSS instances are handled as well as parent/ child relations between device families.

Archive Displays

Good display tools for archive data are essential for data analysis. The tools under construction use the AAL interface to the archive store. A good example for the collaborative approach in the CSS design.

CSS MANAGEMENT

In order to manager CSS instances it is necessary to identify them. A central registry must be in place where the CSS instances can register. This way the CSS management console can query the registry for the running CSS instances and approach each of them. Depending on the management settings it is possible to perform actions remotely. A basic set of remote commands are:

- Setting the update flag to trigger the update process from the update site.
- Query for system and/or application statistics.
- Exchange debugging information with remote CSS instances.
- Last not least it will be possible to shutdown 'left over' (iconized) CSS instances.

To avoid the implementation of a new registry mechanism the XMPP protocol has been chosen to exchange management information. XMPP is an open standard. It is possible to select XMPP client/ server implementations from a wide range of open source implementations. Currently a Wildfire-xmpp server is in operation at DESY. It can be accessed also as a (CSS) chat server (it's basic functionality) from: krykxmpp.desy.de.

NEW DEVELOPMENT STRATEGIES

We are entering new frontiers with the design and development of the Control System Studio. Facing the fact that the in-house resources would not be sufficient to design and implement a complete application suite from scratch, we decided to integrate external expertise from companies and universities into the whole process from the very beginning.

Collaborative Development

A basic requirement for a collaborative development within a wide spread collaboration is the access to all common resources. A cvs repository provides free access to all program sources. Local and remote developers can get access to existing programs and add their developments into the repository. Documentation, a tracking system including the story cards for the agile development process are stored in a collaborative development server based on CodeBeamer [3]

Wizards for easy CSS-Component Development

CSS developments can and will not be driven only by one site. An active collaboration around CSS developments is the final goal. To achieve this goal it is necessary to ease the learning curve for developers. CSS wizards will be in place to help understanding the CSS structure and the essential interfaces.

Agile Programming Strategies

As mentioned above, part of the core developments are carried out together with the University of Hamburg and associated partners. In the course of this cooperation we take the chance to learn and implement also new programming strategies. Agile programming is a technique that originated from eXtreme programming. Requirements are defined in so called 'story cards'. These cards comprise required functionalities, the priority and an associated cost factor. Throughout the development process theses cards are revised on a regular basis. As a result one will create products which a runable from an early stage on while functionality is continuously added.

OUTLOOK

The CSS developments have left the conceptual phase. Core applications are under development as well as utility and management applications. The CSS collaboration is slowly growing. Wizards for newcomers to CSS will help getting up to speed. The collaboration with universities and subcontracted design and programming tasks with industry make CSS a very lively and interesting project.

A release of CSS core plug-ins as well as some basic applications will be available end of 2006. We are looking forward to a fruitful collaboration.

REFERENCES

[1] The web pages of the CSS project: http://www.cs-studio.org.; http://css.desy.de

[2] Emma Shepherd, Minutes of the CSS Core Design Meeting, DESY August 7 – 11 2006, http://css.desy.de/content/e198/e253/e321/index_eng.html.

[3] CodeBeamer – Collaborative Development Platform http://elogbook.desy.de:8181

AUTOMATICALLY CONFIGURED CONTROL SYSTEM USING COMPACT PCI SYSTEM

Y. Furukawa and T. Ohata, SPring-8/JASRI, 1-1-1 Koto, Sayo-cho, Hyogl, Japan

Abstract

An automatically configured control front-end has been developed using compact PCI system combined with Linux hot-plugging function. After selecting and inserting control board into the system and powering on, the system configured automatically and load not only device drives but also control application programs that have to running on the front-end. Application programmers for back-end (client computers) can access the front-end system with ONC/RPC call or simple socket using ready-made libraries or applications without manual configuration of the front-end. The system was successfully tested at the BL01B1 in the SPring-8, which is dedicated for XAFS experiments, to readout a 19-channel SSD with newly developed counter boards. The system is useful for users who are not familiar to the detail of the control system. We are now developing the system adapt to real "hot-plug" system, which provides dynamic configuration ability to the system. Users can change boards and access with simple method in spite of the system still running.

INTRODUCTION

In most synchrotron radiation facilities, many different kind experiments are performed. At SPring-8, for example, around one thousand experiments are performed in year. These experiments spread to wide region of science and technology. Requirements for experimental controls and data acquisitions are also varied from an experiment to other experiments. It becomes difficult task to adapt control system to the frequently changed experimental condition without control specialists, because most beamline scientists and technicians are not familiar to the control system. To modify the experimental station controls flexibly, automatic configuration control system is required.

We have developed automatic configuration control front end with Linux operating system and compact PCI bus system because there are useful tools for automatic configuration for the Linux-OS such as “udev” and “Linux-hotplug”. Compact-PCI brings hot-plugging ability, which enables reconfiguration control system without powering off the system. The automatically configured system loads not only device driver for inserted board but also application programs that controls the board. Users can use the control front-end only inserting control boards, starting-up the system and only sending simple control message via network.

One of the most required I/O board for synchrotron radiation experiments is a counter board. There are many counter boards are available in the commercial market, however they are not suitable for photon counting experiments because of the general purpose design of these boards, i.e. the input connector is not compatible with analogue output of the NIM and/or CAMAC modules. The connection box or conversion cables are required. This spoils portability of the counter system. In addition, it is difficult to realize gate controls and internal timer using a generic counter board. We developed compact-PCI counter board useful in synchrotron radiation experiments and developed it device driver compatible with Linux udev/hot-plug system. Combined with the automatic configuration system and counter board, we have built-up a portable counting system. Only installing required number of board and powering on the system, users can perform their experiments.

SYSTEM DESCRIPTION

The Linux udev system set up device special files for installed devices properly. The Linux hot-plug system configures the system using user space application written in shell script. In the booting process of the Linux kernel, PCI subsystem in the kernel initializes the PCI bus and search PCI devices in the bus. The hot-plug calls a script related to the device found by the Linux PCI subsystem, and notify the device minor number using a environment variable [1]. We have defined a new “class” [1] which contains “class drivers” which describe the individual device information such as device major/minor number, etc. The Linux kernel core exports this information to the user space via “sysfs” pseudo file system and executes an udev application. The udev application makes a device special file referencing the sysfs. Then the udev executes a hot-plug script which name is “<class name>.agent”, and the script starts control application programs then user can access the boards using the applications.

We have installed the MADOCA [2] framework as a control application, because the MADOCA is a standard framework of the SPring-8 control system and because it is flexible and scalable. The hot-plug agent script starts the Equipment Manager Agent (EMA) which controls the installed and configured by the Linux udev/hot-plug system and when the script is firstly called it starts other control framework programs, such as the Message Server (MS) and the Command Interpreter (CI).

COUNTER BOARD AND DEVICE DRIVER

We developed a new counter board, specifications as shown in the Table 1. For compactness, we chose 3U height compact-PCI form factor. The board has four channel counter input and one input/output selectable gate control. These input/output terminals are lemo connectors.

The boards were made by Arkus inc [3], and put on a catalogue as coded AxCPCI3901.

Table 1: Counter Board Specifications

Number of channel	4ch
Maximum counting rate	200MHz
Counter length	32bit/ch
Timer resolution	1MHz/100MHz
Timer length	32bit

The device driver provides primitive functions to access the control registers of the board. Combined functions such as staring/stopping counting, setting gate time, etc. are programmed into the application program (EM). The device driver has 8000 words internal buffer for each channel to store the counting result to obtain millisecond order interval data taking. An interrupt handling routine stores all counter read out into the internal buffer at every interruption. This realises multi channel scaler (MCS) mode useful for quick XAFS experiment described later.

The device driver assigns a rotary switch setting on the board as a device minor number, so users assess the correct board with device special file. Lacking this function, the device minor number will become independent to the board configuration. For example, if the minor number were assigned by the system finding order, the device minor numbers were depends on the installed slot number.

In an initialization function, the device driver creates a "class" axpci3901 and in a device initialization function, the driver crates class drivers "axcpc3901_<n>" where <n> is a device minor number. The Linux udev system generates a device special file "/dev/axcpci3901_<n>" automatically.

PERFORMANCE EVALUATION

Interrupt response

Interrupt response is important to realize the MCS mode measurement. We measured interrupt response time using counter function. We used the Interface Inc. CPU board CTP-PM11A3F which had 1.1GHz Pentium-M processor with 256Mbyte main memory and 20GB HDD and Debian/GNU Linux 3.0 was preinstalled. We upgraded Linux kernel to 2.6.16, because default kernel (2.4.x) was not support udev system. We also installed the device driver described above. 10 counter boards were installed for the evaluation.

10MHz master clock was feed to two channels of a counter board. One counters stopped by an internal gate and the other still counting, the difference between these counters in obtained in the interrupt handler indicates response time. The measured response time distribution is shown in the Figure 1. The measured mean response time is about 5µs, it is short enough compared with the quick-XAFS measurement period longer than 5 ms.

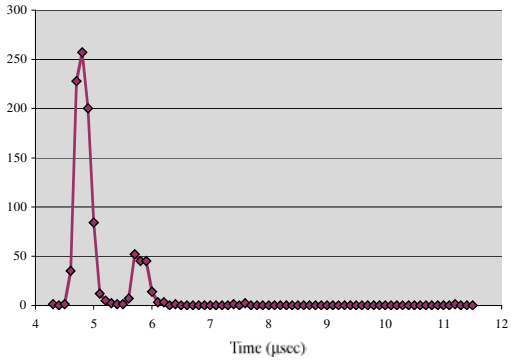
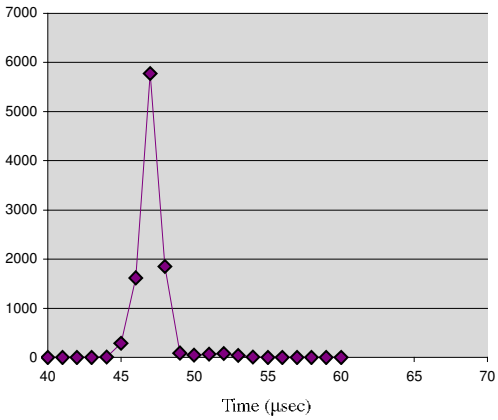


Figure 1: Measured interrupt response time distribution for 1,000 times trial.

Data transfer rate

For quick-XAFS experiment using 19 elements SSD, the data transfer rate from counter to internal buffer is also important factor. Measurement procedure was same as the interrupt response measurement, but we installed 10 boards and measured count difference between the first channel counts of the first board and the final channel in the 10th board. The interrupts were generated by the gate close event of the first channel. Differences include interrupt response time and 40 channel readout time. The measured result shown in Fig. 2. The data transfer time calculated from the measurements was about 1.2 µs, which



was also short enough for the quick-XAFS experiment.

Figure 2: Measured 40ch data transfer time from counter register to internal buffer for 10,000 times trial.

APPLICATION

As described in the previous section, one of the important applications of the system is quick-XAFS experiment, especially for the fluorescent mode. In the quick mode measurement of XAFS, a monochromator is continuously scanning an X-ray wavelength and measures fluorescent X-ray from specified element in a sample with

constant intervals. Typical time intervals of the measurement are 5ms to 100ms. For a dilute element measurement, wide solid angle detector is required. In SPring-8, 19 element SSD is used for the quick-XFAS experiment. For dead-time correction, total counts have to be measured and to obtain the fluorescent yield, single channeled counts have to be measured, so the two counters required for each element, 38 counters are necessary for the 19 element SSD. We installed 10 counter boards to the system and measured Se fluorescent signals with 6ms, 60ms and 120ms intervals at the BL01B1 (XAFS beamline) at the SPring-8. In the figure 3, result for 120ms interval is shown. The result shows the system works well for the quick-XAFS experiments. For the setting up of the experiments, we only wired the counter output to our system and booting up it. We could start measurement easily.

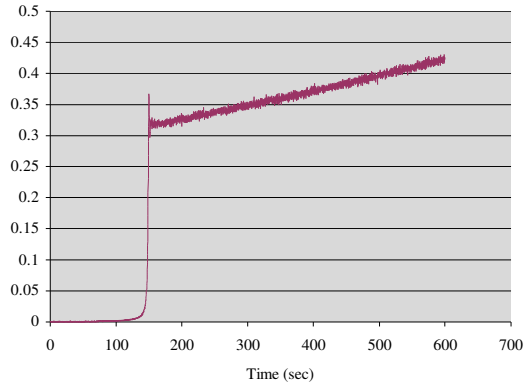


Figure 3: Se fluorescent mode quick-XAFS result with 19 elements SSD and developed system.

SUMMARY

We developed auto-configuration control front end system with Linux udev/hot-plug system on a compact PCI and developed compact-PCI counter board. The system was applied for quick-XAFS measurement successfully and made it easy to built up user required control system. We are now studying an actual hot-plugging of the devices on the compact PCI bus. It will bring more flexibility to the control system because the system can be modified dynamically while the system under running.

ACKNOWLEDGEMENT

For the quick-XAFS measurement, Dr. H. Tanida and Dr. T. Uruga helped us gratefully.

REFERENCES

[1] J. Corbet, A. Rubini and G. Kroah-Hartman, “Linux device drivers, third edition”, O’Reilly, 2005

[2] R.Tanaka, et., al., Proc of ICALEPCS’95, Chicago, USA, (1995) p.201

[3] http://www.arkus.co.jp/

THE COMMON FIRMWARE PROGRAMMING INTERFACE FOR FIELDBUS RELATED PROJECTS AT PETRA III

P.K. Bartkiewicz, S.W. Herb, DESY, Hamburg, Germany.

Abstract

The PETRA II storage ring at DESY in Hamburg is currently used as a pre-accelerator for the HERA collider. From 2009 on, after a major overhaul, it will run as a high brilliance storage-ring-based X-ray source, PETRA III. The new design requires a modern, flexible, and easily configurable control system which will permit the integration of commercial controllers with in-house developed hardware. This document focuses on the fieldbus subsystem and the common application interface for newly designed microprocessor–based hardware, explains why the CAN fieldbus standard was chosen, how the CANopen application layer has been implemented, and how the subsystem will be integrated with the rest of the control system.

INTRODUCTION

The most popular fieldbus at DESY, SEDAC, invented 25 years ago, can no longer compete with modern industrial fieldbuses. Although SEDAC was a very successful data link solution during many years, it lacks important features present in many industrial fieldbuses.

Analysing requirements of the control system for the PETRA III, the high brilliance storage-ring-based X-ray source being built now at DESY, we decided to establish a new hardware development standard for next designs, which unifies the communication on fieldbus level, hardware interfacing to fieldbuses, communication application software layer for device firmware, and fieldbus management and configuration tools. Standardization has to cover:

- The physical and data layers of the fieldbus, and interfacing to the bus.
- A software source code library, representing the application layer of nodes firmware.
- The data exchange mechanism between application specific code and the communication code.
- Automatic management of the configuration of individual nodes.
- Connection to the higher levels of a control system, in our case to the TINE servers.

For the fieldbus several requirements were specified. We need a serial bus, which may be used for short distances in order to connect modules inside a single rack as well as for middle-range (up 100 m) distances, connecting stand-alone modules, such as power supply controllers, vacuum pump controllers etc.. There should be a possibility for device hot plug and unplug, that nodes can be added to or removed from the network without affecting the communication of other nodes. There must also be the possibility for communication participants to detect the plug or unplug of a node. We require a multimaster capability, because the fieldbus communication participants must be able to take over the ownership of the bus in order to generate requests for certain data from other nodes, broadcast node-specific data, and send urgent error or warning messages. In order to achieve high reliability there should be a built-in error detection mechanism. The selected standard should be popular and commercially used, to guarantee availability of fieldbus hardware suppliers - it is important for future developments that manufacturers of the silicon interface chips will support the fieldbus standard for a long period of time. In order not to complicate inter-institute technological cooperation it must be popular, or at least recognized, by other high energy physics laboratories,.

Designers of the fieldbus node hardware should be offered a wide spectrum of interfacing hardware: various microcontroller types with the fieldbus support functionality, stand-alone peripheral chips or IPs for FPGA developers. The firmware part, responsible for the fieldbus communication, should be standardized and available as a 'C' source code stack, in order to permit implementation for various types of microcontrollers.

FIELDBUS PHYSICAL AND DATA LINK LAYER

Among various fieldbus standards the CAN (Controller Area Network [1]) standard looks especially appealing. Although the scope of this document can not cover the entire description of the CAN protocol, some of the major features should be emphasized.

Major features of CAN

CAN offers a multi-master, real-time serial bus architecture, with message-based transmission protected by a very reliable error detection mechanism (message CRC with Hamming Distance of 6, bit stuffing, message consistency check). A corrupted message is automatically retransmitted. The protocol defines the behavior of a node which detects a failure, so the risk of bus traffic disturbances caused by the faulty node is reduced. The principle of node communication is based on broadcast telegrams, so implementation of alarm messages or important process data exchange messages is easy. The message identifiers also serve as priority indicators for non-destructive bus arbitration. The number of nodes is limited only by the physical layer (usually up to 128 nodes). The flexible choice of transmission speed versus bus length offers bus speeds up to 1 Mbit/s for distances up to 40 m, and permits data exchange for distances up to 1000 m with a speed of 50 Kbits/s. Most of the major chip manufacturers provide silicon solutions encapsulating the CAN protocol features. A very wide commercial offering

includes 8-, 16- and 32 bit microcontrollers with one or more built-in CAN interfaces and stand-alone CAN controllers with convenient interfacing to the host logic, such as microprocessors or FPGAs. In case of FPGA based applications the alternative of using IP CAN cores is also available.

Generic fieldbus interfacing hardware

Currently our CAN-related projects for the PETRA III storage ring include designs for the Freescale microcontrollers with built-in CAN controller (16-bit HCS-12 and the 32-bit Freescale ColdFire 5282) and Altera-NIOS FPGA with an external Philips SJA-1000 stand-alone CAN controller. Since the resources of the selected microcontrollers, including internal program memory, ADCs, timers, and serial and programmable I/O ports, are sufficient for future projects, these microcontrollers will play a role as default interfaces between fieldbus and measurement or control applications. For more advanced projects, requiring faster data processing or complicated application logic, the Altera-NIOS with an external CAN controller will be used.

Current applications

There are several designs being made now with the use of our generic fieldbus hardware solution: the Vacuum Controller project based on HCS12, the Magnet Power Supply Controller, and the CAN bus to SEDAC translator (in order to utilize some of the existing Power Supply Controllers), based on ColdFire 5282.

APPLICATION LAYER

Why do we need a generic Application Layer?

CAN as a standard brings very low level functionality, corresponding to the layers 1 and 2 of the well known 7 layer Open System Interconnection model; it offers a reliable mechanism for broadcasting messages containing up to 8 bytes of data, but does not specify how these bytes will be used. It also does not define how to use message identifiers or how to send data blocks longer than 8 bytes.

For PETRA III projects we need a generic set of services which support the automatic configuration of CAN fieldbus nodes (including a possibility for remote firmware upgrade), a standardized message identifiers allocation scheme, a mechanism for node connection and disconnection event detection, and advanced error messaging: services belonging to the Application Layer of the OSI model.

Why CANopen?

As was mentioned earlier, the selection of application layers should be limited to well known industrial standards. Currently the most popular standards are CANopen, DeviceNet, SAE J1939, and CANKingdom. CANopen [2],[3] fits very well to our requirements. Proprietary devices can be combined with CANopen devices into one network, and the implementation is simple and scalable to needs; there is a large set of optional functionality but only a small set of mandatory functionality, so even very simple microcontrollers can be used for CANopen compatible projects. The only hardware requirements are one timer with millisecond resolution and one CAN interface. The software stacks are available for a reasonable price, or even free (open source project). This is also the standard claimed to be the most popular high level protocol. It is very popular in Europe, so many “plug & play” devices are available “off-the-shelf”. The technology is open and does not require any payment or license fees. Additionally the concept of the Object Dictionary with the associated EDS (Electronic Data Sheet) makes updates and maintenance easier, and DCFs (Device Configuration Files) help with automatic configuration of fieldbus participants.

Implementation

In order to get a reliable implementation of the CANopen protocol for selected hardware platforms, we decided to buy a commercial software stack from the Vector Informatik GmbH company [4]. So far the stack has been ported to the Freescale processors; the port for the Altera-NIOS is planned to be done soon. For separation of the application firmware from the CANopen stack, a set of callback functions were created. Developing the firmware application now consists of filling a very limited number of functions with user code. There are two sorts of such functions. Simple callback functions are called once, to obtain from the user information such as selected bus speed or node ID needed for a proper CANopen stack initialization,. There are also callbacks called once to notify the user code that the internal state of the stack has been changed. A second group of callback functions are called cyclically, when the CANopen stack is idle, or just after a single CAN message has been processed. These cyclical callbacks are the places where the firmware developer puts his working code performing ADC or digital input line readout, steering output lines, or doing necessary calculations. It is worth emphasizing that the firmware developer does not write a single line of code related to the fieldbus data exchange; he concentrates on data collection and processing, and the data transfer is done automatically by the CANopen stack. Which data are transmitted and what is the transmission triggering mechanism is described by the few entries in the Object Dictionary structure. The developer is offered some macros, which makes such Object Dictionary manipulations easy.

Firmware development for CANopen compliant node requires carrying out three steps:

- Make ‘global’ all variables, which should be transferred over the CAN bus
- Using appropriate macros to connect them to the Object Dictionary, and select the transmission triggering mode.
- Fill the skeletons of the callback functions with your code.

In order to integrate the node with the control system, one must perform a fourth step:

- Create an EDS file, which describes the data available from the CAN bus and the data transmission modes.

The EDS should reflect the entries in the Object Dictionary structure, so for bigger and more complicated projects the use of graphical tools, also commercially available, is recommended. Such a tool usually leads the developer through the process of adding entries to the Object Dictionary and defining the data transmission modes for selected variables, checks consistency of the Object Dictionary, and automatically generates the EDS file. For our development work we use the CANerator program from the Vector Informatik GmbH.

CONNECTION TO THE CONTROL SYSTEM

The fieldbus and its nodes are part of a larger control system; measured data should be accessed by servers, processed and delivered to the users, and nodes should be configured for work with changing requirements. We want also to provide a standardized solution for integrating CAN bus and the CANopen protocol into our control system.

Hardware and operating system

As a hardware platform for the servers connecting to the fieldbus we decided to use the PC104, a well known and popular industrial standard supported by many manufacturers. PC104 can be easily extended by stacking additional cards, including CAN cards. Since the PC104 is a PC-compatible platform (80x86 or Pentium processors), it is possible to use Linux as an operating system

Our current solution consists of a PC104 card produced by the Kontron AG [5] and the dual channel CAN produced by the PEAK-System Technik GmbH [6]. The Peak company delivers Linux drivers for their CAN cards. The stack of PC104 and CAN card is adapted to fit in 3U Euro Crate. There are no mechanical hard drives; we use a 32 MB flash disk. The Linux distribution is ELINOS, an embedded Linux distribution from SYSGO AG [7]. ELINOS comes with a package of configuration tools which enable scaling and quick setup of the kernel components, and makes transfer to the flash disk easy.

To increase reliability of the PC104 operations, especially in harsh environments, the hardware watchdog on the PC104 board is activated. To avoid problems with potential disk data corruption caused for example by power outages, the flash disk partition is normally mounted 'read-only'.

TINE server for fieldbus connectivity

The control system for PETRA III will be TINE [8], the most popular control system at DESY. Since TINE supports all popular operating systems, the server code is easily ported to ELINOS. TINE's footprint is small enough to fit into the system equipped with only the small flash disk. Programmers who use the embedded TINE server can choose between using CDI plugs (Common Device Interface [9]) or writing own CANopen specific interface. In both cases they will deal with the very well known TINE server structure, with its standard logic and configuration files.

Besides connection to the rest of the control system, our current implementation now offers very limited fieldbus management, which however can not be treated as a generic solution for future developments.

We need some additional functionality:

- detecting whether the node needs to be setup due to reboot or having just been added
- carrying out the after-boot configuration process of fieldbus nodes, which usually requires the parsing of individual DCFs and sending individual settings to nodes, or connection to an external data base which stores such settings data
- large data block transfers for nodes which offer remote firmware upgrades over the CAN bus
- generation of some CANopen specified messages, such as NMT messages, SYNC, or time stamp.

We are now investigating whether we should buy commercial code, or whether writing own CANopen master components will give us better integration with the TINE environment.

REFERENCES

[1] Wolfhard Lawrenz "CAN System Engineering. From Theory to Practical Applications" Springer-Verlag New York Inc.1997

[2] Olaf Pfeiffer, Andrew Ayre, Christian Keydel "Embedded Networking with CAN and CANopen", RTC Books 2003

[3] http://www.can-cia.org

[4] http://www.vector-informatik.com

[5] http://www.kontron.com

[6] http://www.peak-system.com

[7] http://www.sysgo.com

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

[9] Philip Duval, Honggong Wu "Using the Common Device Interface in TINE" Proceedings of this conference.

UI-ORIENTED APPROACH FOR BUILDING MODULAR CONTROL PROGRAMS IN VEPP-5 CONTROL SYSTEM

D.Bolkhovityanov*, The Budker Institute of Nuclear Physics, Novosibirsk, Russia

Abstract

Specialists combining good programming skills with perfect physics are usually in deficit. So, often physicists prefer to write control programs themselves.

But, besides the "meat" — application control logic — a program must include some routine "spells", like standard GUI programming, conversation with data server, etc. These lay extra burden on a developer, and, while similar in most applications, these "spells" are hard to move into a separate library, since they are usually too interwined with application logic. However, such separation is highly desirable: common parts can be implemented bullet-proof and feature-rich by professionals, leaving only tiny specific parts for particular application developers.

So, a modular plugin-based architecture was developed for VEPP-5[1] control system, which separates program data description from its implementation. Thus it allows to store this UI-oriented, but GUI-free description in a database. This description is also used by health monitoring, data archiving and web-publishing tools.

COMMONLY USED APPROACH

Large portion of most facilities' control systems consists of "screen instruments" — meters and control knobs, which are directly mapped onto ADC inputs and DAC outputs.

To simplify development of applications which are constituted entirely of such screen instruments, most control systems provide so-called "display managers", which take care of interaction with operator, thus eliminating a need to write any code. These are `MEDM`[2]/`dm2k`[3] in EPICS, `ddd`[4] in DOOCS, etc. VEPP-5 is no exception here — its control system CX[5] includes such "screen manager", called `chlclient`.

"Display managers" generally take some descriptions, which specify lists of display components, including their kind (text fields, menus, graphs, etc.) and positions, plus mapping of those components onto hardware channels, and build user interface "screens" accordingly.

VEPP-5 Specifics

VEPP-5 is under construction now, changes are frequent, so, instead of "visual", "symbolic circuit" view, a "system-centric" approach is used: one window shows vacuum status of the whole facility, another one is devoted to thermostabilization, etc.

* D.Yu.Bolkhovityanov@inp.nsk.su

MEDM/dm2k, DDD and many others use "canvas" model, which allows a user to put components into arbitrary places of a control window. VEPP-5/CX's main approach is grid-based (see Fig.1): components ("knobs") are laid out in a regular grid, with row/column labels added if necessary; "towers" of rows (called "stairs") are wrapped into several sub-towers if required. *("Canvas" model is also supported, but is little used yet.)*

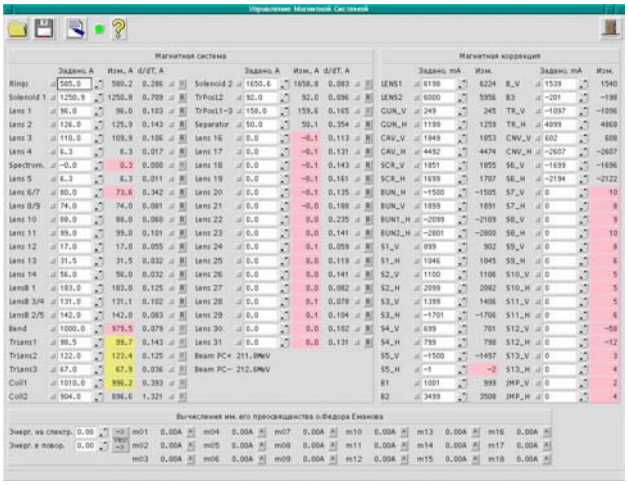


Figure 1: A typical VEPP-5 control application — linac magnetic control. There are 3 screen "elements", upper two are two-"tower".

CX applications' screens consist of 3 layers (see Fig.2):

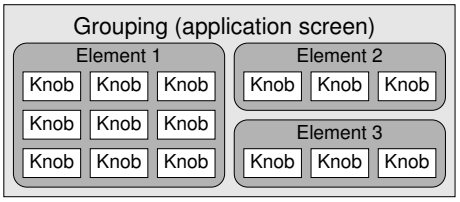


Figure 2: 3-layer structure of applications' screens.

1. Display knobs themselves.
2. Knobs are grouped into *elements* — also called *containers*, which place "streams" of knobs into grids, labeling and wrapping as required.
3. A set of elements is called *grouping*, which essentially is an application's screen. A grouping performs simple automatic layout of elements — by rows or by columns, wrapping as needed.

Elements can be nested — any cell in a grid can be occupied by a sub-element instead of a knob. That sub-element can also contain sub-elements, and so on. This allows to build display hierarchies of arbitrary complexity.

So, description of an application's screen is a *tree*. And these tree-like descriptions can be stored separately.

THE CHALLENGE

But: if something more than a trivial display of hardware channels is required, notably one of the following:

- Some computations beyond trivial arithmetic.
- Data must be displayed in some unusual way, so that built-in components aren't sufficient.
- One-to-many relationship of control channels: when change of some display knob must result in modification of *several* hardware channels.
- Some specific data processing, feedback, etc.

— then "display manager" can't be used and such application must be coded "by hand".

VEPP-5 Specifics, continued

CX supports regular, *scalar* channels and more complex, *array* channels (fast ADCs, CCD-cameras, etc.; those are called "big channels" in CX) in a different manner, due to a different nature of tasks.

Moreover, in any control system, array channels often have to be displayed in some specific way — just a graph/histogram isn't sufficient.

But it is hardly possible to equip a "display manager" with more or less unified display components for all flavors of array channels — display requirements are usually too diverse, and often a specific way of data display is the essence of a respective program.

So, all CX applications, dealing with "big channels", had to be coded individually.

THE SOLUTION

The Idea

A similar problem had existed in many areas; the most widely known example is WWW — how to handle multimedia content — and that problem has got an adequate solution: *plugins*[6]. A browser doesn't know what to do with, e.g., Flash animation, but it looks at information, describing `.swf` file as "application/x-shockwave-flash", to select appropriate module, called plugin, which does know how to display such a file.

A "display manager" can work in a same manner: when it encounters a component, which isn't tagged with one of built-in types, a search for plugin is performed, and that plugin is used just like built-in types.

Different "Classes" of Components

The initial goal was to enable placement of diverse user-supplied visualizers of array channels into "display manager"'s screens. But making other kinds of display components also "pluginizable" gives many advantages. These "classes" include:

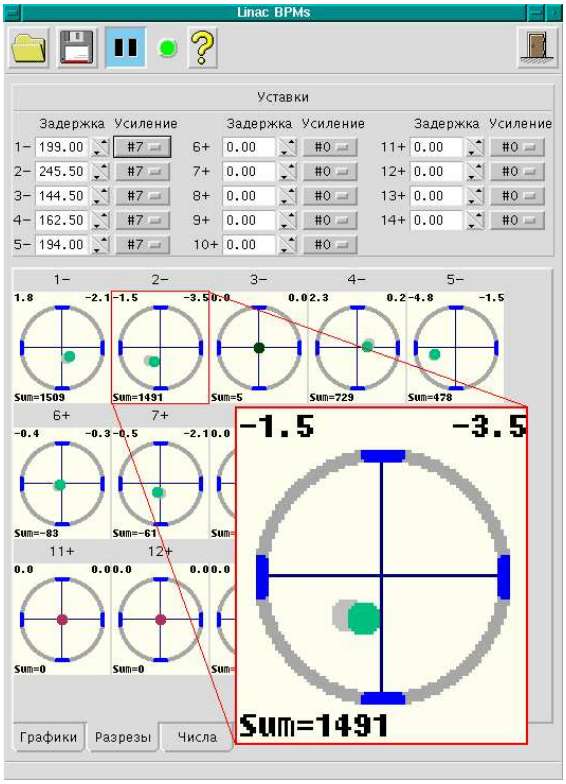


Figure 3: Example of a plugin-based application: BPM display. It is a mixture of standard knobs (top element) and plugins (bottom one). Pale-gray circle beside the beam position is a "shadow", showing previous position. The bottom element is another example: "tabs" are provided by container-plugin.

- Scalar channels. So, new kinds of knobs (like "wheel-switch") can be added easily.
- Array channels.
- Containers. For example, a "worksheet with tabs" (see Fig.3). Or, a "virtual damping ring", which automatically lays out its sub-components in a ring.
- Decorations — data-less components, which just appear.
- Groupings. So, a generic "vacuum system" grouping would enable switching between a regular bar graph view and a detailed numeric view of all ion pumps' data (which is frequently requested by technicians).
- User-type components — when the task can't be reduced to simple on-screen work with one scalar or array channel; or requires some specific processing or feedback; or just something else, not falling into one of the above classes.

"Fallbacks"

There is an important difference from e.g. web browsers: nobody except Flash plugin can handle Flash files, but since there are only few "classes" of components in control system's displays, a *fallback mechanism* can be used. Text field would be a reasonable replacement for any scalar-

channel-plugin, a graph can be used for array channels, etc.

Two "parallel" trees

First, the *data* tree is created, which contains complete information about used channels (references to hardware channels, limits, names, labels, etc.). Then the *display* tree is "rendered" accordingly with that data.

In cases when only data or just a cumulative status is required (see "Additional Bonuses" below), creation of display tree can be omitted — the data tree is completely functional in its own.

Some Details

CX "display manager" — `chlclient` — is just a mere frontend to a set of GUI libraries: libChl (stands for "CX high-level"), providing a general application functionality; libKnobs, providing knobs API and a default set of display knobs; a tree-management library, libCdr, which can also be used in non-GUI applications (see "Additional Bonuses" below).

Internally, there's very little difference between standard components and plugins. Both make use of the same knobs API, and when libKnobs searches for appropriate display component, it just looks through *several* tables, instead of one.

In fact, the "canvas" display model is implemented as plugins in CX: the "canvas" itself is a container-plugin, while various decorations (rectangles, ellipses, lines, etc.) are decoration-plugins.

ADDITIONAL BONUSES

Since display channels' specifications are separtated from code and are stored either in files or in a database, these descriptions can be directly used by various generic control system tools. These include:

- "Control center", which works as control system's start menu and as health monitor (see Fig.4).
- Web-publisher.
- Data archiver.

So, any control application (which is usually a reflection of some subsystem of a facility) automatically becomes fully integrated into control system's environment.

VEPP-5 USE

As more and more VEPP-5 subsystems are put into regular operation, demand for various specialized, not-just-a-grid-of-text-fields applications, grows.

In the last two years the majority of such applications in VEPP-5 control system are implemented in a plugin-based fashion. This quickened the development and made the control system more flexible.

Figure 4: CX "control center" — CX-starter. Round leds in "Srv" column show status of appropriate device servers, square leds in "?" column signal channels' status (cumulative for a subsystem).

REGARDING NOVELTY

While tree-like descriptions of control screens aren't new — such approach has wide use, from Visual Basic to LabVIEW — there are two significant differences in the approach presented above.

First, a *data* hierarchy exists separately from a *display* hierarchy. These trees are either "parallel", or display tree doesn't exist at all.

Second, plugin-architecture with "fallbacks", where plugin displays can be provided for scalar channels, array channels, decorations, containers and "generic" user/application-specific components, enables to turn *any* control application to a unified form.

CONCLUSION

UI-oriented approach for building modular control programs proved to be very useful. It significantly eases development of control applications, and makes the whole control system more structured and modular. This approach will get further wider use in VEPP-5 control system.

REFERENCES

[1] M.Avilov et al., "Status of Work on the VEPP-5 Injection Complex", Atomic Energy, Volume 94, Number 1

[2] "EPICS MEDM: Motif Editor and Display Manager" `http://www.aps.anl.gov/epics/extensions/medm/`

[3] Thomas Birke, "dm2k" `http://www-csr.bessy.de/control/SoftDist/dm2k/`

[4] Kay Rechlich, "ddd: The DOOCS Data Display" `http://tesla.desy.de/doocs/doocs_gen/ddd.html`

[5] D.Bolkhovityanov et al, "Evolution and Present Status of VEPP-5 Control System", Proc. PCaPAC'2002, `http://www.lnf.infn.it/↩ conference/pcapac2002/TALK/MO-P15/MO-P15.pdf`

[6] "Plugin", Wikipedia, the free encyclopedia `http://en.wikipedia.org/wiki/Plugin`

A DEVICE SERVER GENERATOR FOR CONTROL SYSTEMS

J. Wilgen, P. Duval, DESY, Hamburg, Germany

Abstract

The device server generator is a wizard-like tool which substantially minimizes the efforts needed to build device servers used in control systems. It defines a common model for the device server and its interaction with the underlying control system protocol. The current prototype focuses on Java-based device servers using the TINE protocol [1]. Interface specifications can be defined with a GUI, and TINE-specific code is generated automatically.

INTRODUCTION

The device server generator has the purpose of standardizing device server programs and interfaces and simplifying the work of device server programmers. Up to now, device servers were individually stylized. The existing code generators [2] created device server code which served more or less as a template and had to be extended by programmers in order to do something useful. To be sure, this must always be the case. However, in the past considerable knowledge of the device server logic was required to make the necessary extensions. The new device server generator creates a full framework in which only the device logic needs to be supplied by the device programmer. The framework consists of a generic part in form of a runtime library, and a generated part which includes specific classes and TINE-related code.

MAIN FEATURES

Common Logic is Encapsulated in a Framework

A great part of the program logic and structure of device servers can be generalized by a common framework so that programmers do not have to continually supply recurring, similar logical constructions.

TINE-Specific Code is Hidden

A considerable part of a TINE device server consists of code which is needed to implement the TINE interface. The TINE-related code is now generated and hidden in the framework. The programmer need only be concerned with the implementation of device logic and functionality.

Standardized Network Interface

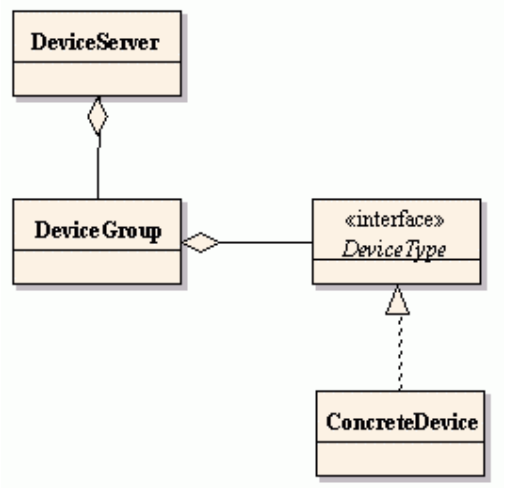
Since the framework is responsible for the server's network interface, a common interface structure can be used. This makes it easier for clients to interpret a server's network interface.

Wizard GUI

A graphical user interface is used to define the server's components and to generate the code. Generated code can be reworked at any time.

THE BASIC MODEL

A device server framework must be based on a structure which is applicable for a majority of cases without being overly complicated. It must also be possible to map this structure easily into a TINE interface. We found that device servers have quite different structures, but most of them can be described sufficiently with the following simple model.



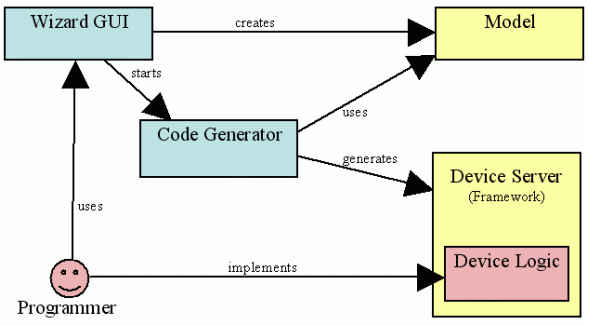
A device server can contain one or more device groups. Every device group contains elements of a common device type which defines the device's interface. Every device in the group must implement this interface. A device interface consists of properties (= methods which input or output data of a simple or pre-defined type) or operations (= more complex methods).

Although it is not possible to map this model exactly to TINE, it can be done if the device server above has a one-to-one correspondence with the device group. A device group corresponds to a TINE equipment module, and properties and operations are mapped to TINE properties.

THE PROGRAMMER'S WORKFLOW

For the specification of the device types and device groups in a device server, programmers use a wizard GUI. The result is a device server model which can be stored independently. From the GUI, a code generator can be started which generates source code which includes parts of the device server framework as well as stubs for the concrete devices which have to be implemented. The

same tools can be used to revise the stored model and re-generate the code later.



LIMITATIONS

Applicability

The device server generator is a suitable tool for many cases but is not a solution for everything. There are still device servers which do not fit into the specified structure, or have very special requirements which cannot be satisfied by a generic framework.

Encapsulation vs. Customization

Since the framework keeps TINE-specific code away from the programmer, it is impossible to customize TINE interfaces directly. The ability to overload TINE properties, handle calls from the local archive or alarm sub-systems differently, make use of device name patterns, for instance, is no longer available.

Standards vs. Flexibility

By means of structures, rules, and simplification, the device server generator makes device server programming clearer and easier for those focusing solely on the device logic. Programmers who want to use the total flexibility which TINE offers will be better off building device servers manually.

CONCLUSION

The device server code generator presented here is currently available only on java platforms. It is targeted to conform to a general set of TINE device server specifications, but is by no means limited to TINE, and can be used with other systems with a similar device naming hierarchy. As a number of TINE features are encapsulated away from the use, they are simply unavailable. This ultimately means that the generated code will satisfy the needs of 'simple' device servers, where there are no demands for complex business logic. Nonetheless, there is generally quite of few of such 'data-acquisition' based server, where the server requirements do not go much beyond acquiring data from, say, a group of temperature sensors, setting thresholds, etc. For such cases, the device server generator is ideal, as the work involved in making the data available to the control system effectively shrinks to zero, freeing the control system staff for the more demanding needs of the complex device servers.

REFERENCES

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

[2] P.Duval and V.Yarygin, "The Use of Wizards in Creating Control Applications", ICALEPCS Proceedings 2001; See also Ref [1].

LivEPICS – An EPICS Linux Live CD for small applications, training and fly tests.

M. Giacchini, G. Bassato, INFN-LNL, Legnaro, Italy

Abstract

EPICS is a software tool-kit originally developed at Los Alamos National Laboratory and Argonne National Laboratory for the control of accelerators and large experiments. Since the version R3.14.1, released in 2002, EPICS was ported to different hardware and software environments and now it is available for many kinds of processors and operating systems.

Despite the installation of EPICS is usually done by a well proven set of automatic procedures, its configuration is not always straightforward for a beginner. To help a new user to get familiar with EPICS tools without installing them on the hard disk, we developed a Linux based live CD that includes most of the EPICS features.

LivEPICS is a bootable CD, which contains a pre-configured EPICS development environment. After booting, the user can access all the utilities required to create a simple control application; at the end of the session he can save his application on a USB mass storage device.

EPICS BASICS

The basic idea underlying EPICS[1] architecture is the implementation of a software bus. Process variables are declared, through a graphic tool, as records of a real time data base. The record properties define the method by which the record is processed: usually, a record is associated to a particular hardware device and processing a record means to call a device driver that acquires the variable and writes it in the data base. Record processing is realized by a software module named IOC. Once a variable is stored in the data base it can be accessed by multiple clients through a network infrastructure called Channel Access. Provided the device support for the hardware already exists, a control system can be designed and tested with no need of writing program code.

The toolkit includes a lot of utilities: i.e, the Motif Editor and Display Manager to generate graphic panels, the Alarm Handler, the Data Archiver, the Channel Access Probe and many interfaces to make available the data base variables to non-Epics applications.

LIVEPICS FEATURES

LivEPICS[2] is a Linux live CD that includes: Epics Base (release R3.14.7), Extensions tools, introductory documents and manuals. It has the complete functionality to develop a small control system, but it is mainly intended for training classes or to monitor and supervise an EPICS network.

The goal of LivEPICS is:

- Allows to use EPICS without installation on the hard disk.
- Automatic setup of environment variables to compile and test new applications from scratch.
- Includes the basic tools (MEDM, VDCT, etc.) with the related documentation.

The iocBaseApplication (the utility that creates the directory structure necessary to develop an application) can be launched immediately after the boot. The OPI tool included in the CD is MEDM (Motif Editor and Display Manager), the alarm manager is AH (Alarm Handler) while the IOC database configuration tool is VDCT[3] (provided by Cosylab). The Channel Access Probe is available to test the status of a record on the network. Asyn and MSI packages allow to create device support applications and medium-sized EPICS DataBases. The CD includes the following documents: Application Developer Guide, IOC Application Building, Record Reference Manual, Channel Access Manual, Channel Access Protocol, State Notation Language Manual.

To develop this live CD, we used, because of its reduced size, Linux SLAX (based on Slackware[4] with kernel 2.6). The file system on the CD is SquashFS, a highly compressed read-only file system that is specifically designed for tiny Linux systems.

In the CD, the standard directory tree of the SLAX Linux filesystem is compressed to a standalone file; all files that constitute the EPICS base tree are installed, compiled and compressed to a module named EPICS-3.14.7-base_Asyn_msi_VDCT.mo not larger than 33MB. This directory structure significantly improves the access time to the files most frequently used in the development process.

CONCLUSIONS

We have created a bootable CD that can be useful for training or monitoring purposes. We will also develop the driver support for a generic binary device connected to the PC parallel port to run a real demonstration of a control application.

REFERENCES

[1] EPICS, www.aps.anl.gov/epics

[2] LivEPICS, www.lnl.infn.it/~giacchini/upload/file.php

[3] R.Sabjan et al., "Visual DCT – EPICS Databases can be Fun", PCaPAC 2002, Frascati, Italy, October 2002

[4] Slackware, www.slackware.com

FAULT-TOLERANT EPICS DIRECTORY SERVICE*

I. Habjan#, K. Zagar, M. Sekoranja, Cosylab, Teslova ulica 30, SI-1000 Ljubljana, Slovenia.

Abstract

Experimental Physics and Industrial Control System (EPICS) [1] is using the Channel Access (CA) [2] protocol to discover and remotely access records and their individual fields (process variables, alarm states, ...). For discovery, the client-side CA issues a broadcast on a subnet, to which an input-output controller (IOC) hosting the record responds with the location (host and port number) of the record. This is a very convenient method for discovery, as no initial setup is required and there is no single point of failure. It does, however, have disadvantages, such as large amounts of network traffic produced and error-prone configuration in networks with several subnets.

In this paper, we describe an approach based on a Directory Service (DS) that addresses these disadvantages while introducing additional features. Among the new features are load balancing, attribute-based discovery, support for redundant channels and extraction of client-server topology in an EPICS system. To avoid the directory service from becoming a single point of failure, it is replicated using techniques developed in the Dependable Distributed Systems (DeDiSys) [3] project, co-founded by the European Union. The directory service can be introduced in a running system with simple reconfiguration of nodes, but without requiring any change to existing implementation. The prototype implementation of the directory service is EPICS V3 compatible, and the design has been made with EPICS V4 in mind.

INTRODUCTION

An EPICS based distributed control system must consist of at least one Channel Access Server (CAS). Usually this is an EPICS process called IOC. The computer running the process is attached to input and/or output devices of technical equipment. This is interfaced using the EPICS database of records. The data is within a record contained in its VAL field and is made accessible via process variables. This provides logic to acquire the value from a sensor or to impose a value to an actuator in a physical processing environment. Such is the association of PV values with the results of the input/output operations. The set of all PVs distributed over several IOCs establish a distributed real-time database of information and control parameters. A synonym for PV is channel.

Channel Access Name Resolution

Channel access is a TCP/UDP/IP based communication protocol used by EPICS. CA allows Channel Access Clients (CAC) to require access to PVs. Client may see and change values or monitor value changes. As PV is referred to by its name, CA needs that name to access data.

To connect to one PV the following search procedure must be used to discover its location (Fig. 1):

1. Client broadcasts a sequence of UDP packets on a subnet.
2. All IOCs receiving the broadcasted message check whether they host the sought channel.
3. The IOC that hosts the channel responds to the client.

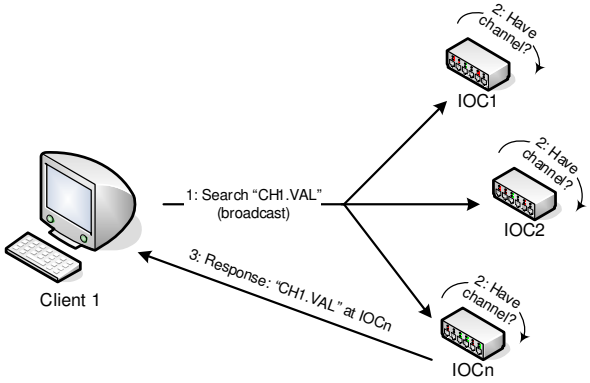


Figure 1: Channel access name resolution.

Even though this is a very convenient method to initiate a communication it is one of major problems with EPICS.

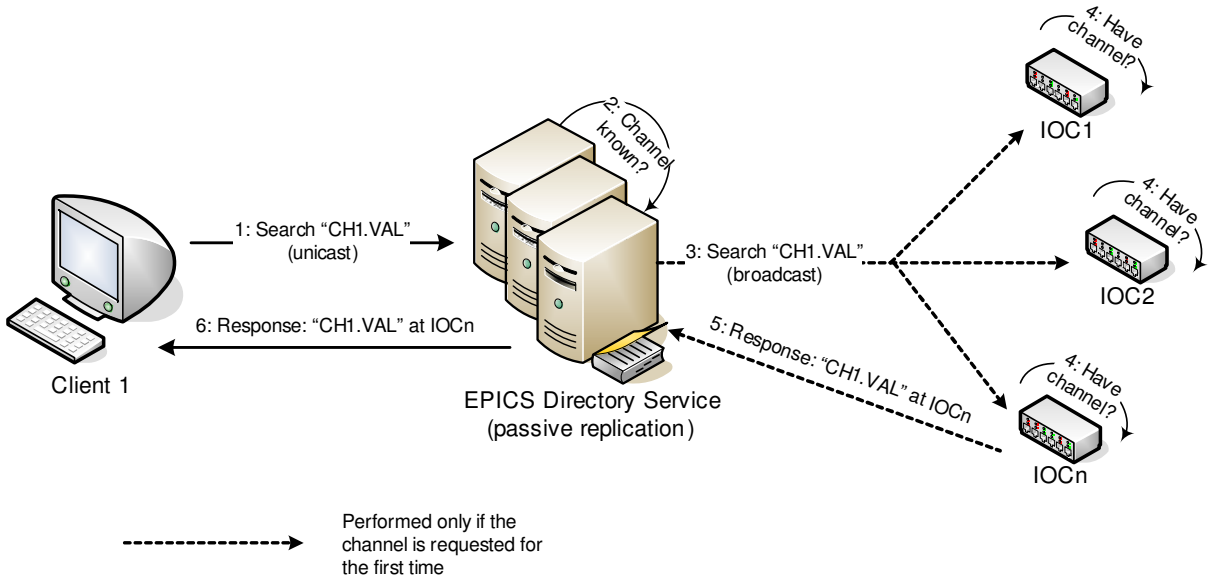
To target multiple IOCs the search requests have to be broadcasted. Broadcast can only be done via UDP and is thus unreliable. To cope with possible message loss (requests or responses) the latter are repeated. The search request itself therefore consists of a sequence of UDP packets (up to a 100, sent over period of 8 minutes). This is carried trough only with non-existent PVs that do not render a response. Nevertheless re-transmission may lead to excessive network traffic – also known as network storms. Each IOC has to check its inventory with every packet. Another thing with broadcast is that it works only on the local subnet.

EPICS environment variables may be used to reconfigure the CA [4]. To override the default broadcast, EPICS_CA_ADDR_LIST may be set to list specific search addresses, also of other subnets. In combination the EPICS_CA_AUTO_LIST must be set to NO in order for the default broadcast not to be included automatically. System reconfigurations require that all clients are properly configured, which can be tedious in large deployments.

This design makes construction of a directory service (also called naming service) possible without requiring modifications to the existing code. Furthermore, a directory service will be possible to deploy by simply reconfiguring the system.

*Work supported by European Union within the Framework Programme 6 under Contract number 004152.

#igor.habjan@cosylab.com



REQUIREMENTS

Currently, EPICS does not feature a directory service. In preparation for the next version, V4, its requirements are discussed in [5]. The principal function required by the EPICS collaboration is the name resolution. And beyond, the following:

- Minimal setup effort: The directory service should configure itself on-the-fly if no configuration is given (e.g., by establishing name—IOC mappings as they are requested using existing name resolution).
- Lookup performance: The lookup performance should be up to an order of magnitude slower for the first lookup, and as fast as without the DS for subsequent lookups.
- Redundant directory services: A hot-standby backup DS is available, which takes over name resolutions in case of primary's failure.
- Bind performance: The performance of a bind can be up to an order of magnitude slower than the performance of a lookup without the DS.
- Scalability: The DS should not have any particular scalability bounds. It should be capable of handling 1 million PVs. The algorithms for binding and lookup should be better than $O(N^2)$.
- Wildcard searches: Clients should be able to resolve channels to IOCs using wildcards. Instead of wildcards, regular expressions could also be used. This will significantly reduce effectiveness of lookups from $O(1)$ to $O(N)$.
- Report on number of channels per IOC: The DS should be able to report how many channels a particular IOC is hosting.
- Redundant IOCs with the same PV: The same PV can be hosted on more than one IOC. The DS is capable of redirecting clients to the IOC most capable of serving a PV (e.g., the active IOC, or the one with least load).

The main purpose of the directory service is to reduce the loads imposed on the IOCs with processing all the broadcasted searches in particular of non-existent PVs. Achieving a more global overview of the system, this could also reduce excessive loads on the network and give feedback on bad application configurations. The main goal is to eliminate invalid PVs. Further value-added services may leverage on the implementation.

IMPLEMENTATION

In EPICS, the name of a channel is encoded as string of ASCII characters. For every channel, the directory service keeps track of all IOCs that host a channel with that name. Typically, a channel is hosted only on a single IOC, however the directory service shall assume that there can be more than one. An IOC is defined by the host and port where the IOC process is listening.

Channels are associated with the IOCs in a simple directory data structure with operations presented in Table 1:

Table 1: Directory operations

Operation	Description
Bind	Associates the name with the IOC.
List	Returns a set of all administered IOCs and further a set of names of all channels hosted by the specified IOC.
Resolve	A set of all IOCs which host a channel with the requested name is returned.
Unbind	Removes the association of a name to the IOC.

Configuration

The EPICS Directory Service is introduced as a channel access server. As such, it is listed in the EPICS_CA_ADDR_LIST of every client (EDM screens, IOCs with links over CA, etc.). This directs the searches to the single address. In most cases the DS will know where the channel is located, or will know with certainty that a channel does not exist. If so, it will immediately reply to the client (see Fig. 2, step 6). If queried for a channel for the first time, it uses standard CA procedure to discover the channel (Fig. 2, steps 3 thru 5). The response message is used to obtain the IP address and port. Binding is created for the requested channel. The client is also notified of the discovered location.

By monitoring the beacon heartbeats the service may discover that an IOC is down or rebooted. If so, it will unbind all channels from that IOC.

FAULT TOLERANCE

Fault tolerance is provided through DeDiSys framework. The framework considers a network that suffers from node and partition failures (e.g. router or network cabling failures) form time to time. With that considered, the aim is to enhance availability. Fault tolerance is achieved by replication.

This fits quite well into the EPICS scenario, because single EPICS DS poses a potential single point of failure. This may be copped by replication:

Several instances of service can exist simultaneously. All instances are active at the same time, listed by clients in their EPICS_CA_ADDR_LIST. In order to further retain network traffic low, processing is handled only by a selected worker instance. Every change is afterwards propagated to other instances keeping them synchronized.

At the time link failures appear, groups of nodes might become separated. Careful positioning of replicas will preserve each group sustained: comprised of number of CA clients and servers with one, preferably several, instances of DS.

The novelty of the DeDiSys approach is that it enables replicas to recover from the failure and to continue to perform normally. Ability to do so is required by the scenario, because EPICS control system continues to operate in need of a DS despite failures. States in partitions evolve independently and for this integrity constraints have to be relaxed. As this was not always the case it is experimented by DeDiSys with primary-per-partition-protocol (P4). By the protocol, further degradation is dealt with similarly, also the loss of a node.

When a node recovers, it obtains latest state from the other DS instances. With reunification of partitions the states of all instances are automatically reconciled to obtain the state of the full system.

CONCLUSIONS AND FURTHER WORK

The EPICS Directory Service talks channel access and is platform independent. Its function, performed as a custom CA server, is to know the name of each PV. It enables a CA client to find a PV, even on another subnet.

Introduction into EPICS based control system preserves the main advantages: distributed, fast, efficient and reliable. Such are also the characteristics of the DS.

Injected with the DS, the control system continues to operate continuously. Reduced is the CPU load on IOCs: the processing on name resolution is removed from IOC-dedicated piece of hardware to a platform independent server machine. As a side affect the network traffic is reduced considerably. Search procedure communication is no longer broadcasted. Name resolution is targeted to a centralized service. Even though centralized, the service does not compromise the system. This is due to its distributed, fault-tolerant architecture.

Directory service is more than a plain name resolution service. It is a directory: a searchable, up-to-date list of all channels in use. And for more conclusive identification of non-existent channels: a list of all non-existent channels, along with which client requested them. Designed with extensibility from the bottom up it features a web service to display the bindings.

Further work includes even further exploitation of the presented features. Redundant channels may be used for channel load balancing and fault-tolerance, by simple redirection of the name resolution. The collected real-time information may be enriched with meta-information, obtained from SQL databases, LDAP servers, text files, etc. Centralized management of an EPICS deployment (e.g., current status of IOCs) is a next step.

Most importantly, compatible with EPICS 3.14, it requires only a simple reconfiguration: no modification of existing clients or servers is needed; no setup is required, as the list of channels is populated automatically.

The prototype implementation of the design is available for download at:
http://www.cosylab.com/Cosylab/EPICS_directory_service

Feel free to try it out. Eliminate your non-existent PVs!

REFERENCES

[1] Experimental Physics and Industrial Control System: http://www.aps.anl.gov/epics

[2] A. Pucelj, “Channel Access: Protocol Specification”, Cosylab, 2004, http://epics.cosylab.com/cosyjava/JCA-Common/Documentation/CAproto.html.

[3] Dependable Distributed Systems (DeDiSys), 6th European Framework project of the Information Society Technologies (IST) priority, under Contract number 004152: http://www.dedisys.org

[4] Jeffrey O. Hill, EPICS R3.14 Channel Access Reference Manual, 2004, http://www.aps.anl.gov/epics/base/R3-14/6-docs/CAref.html.

[5] EPICS V4 Name Server (Wiki discussion), http://aps.anl.gov/epics/wiki/index.php/V4_Name_Server.

JAVA DEVICE API AND COSYBEANS IN THE GSI CONTROLS SYSTEM

G. Fröhlich, K. Höppner*, U. Krause, V.R.W. Schaa, GSI, Darmstadt, Germany
I. Križnar, M. Pleško, J. Bobnar, Cosylab, Ljubljana, Slovenia

Abstract

Rebuilding central parts of the GSI control system will replace proprietary components by well established standards. Network access from applications to the front-end device servers is the core of the renovation. The former home-made communication protocol is replaced by an access based on CORBA. This broadly supported standard opens the GSI control system, formerly restricted to OpenVMS with software mainly in procedural languages, to a broad range of operating systems and languages. A project was started to rewrite GUI controls in Java. Best available solutions, like Abeans, CosyBeans and other community driven solutions are evaluated in the process. The flexible operation of the GSI facility which handles several beams in parallel on a pulse-to-puls basis requests thorough consideration of the data access and device model API. We will report about the experience and will present the derived status.

GSI CONTROLS SYSTEM

Architecture and Software Structure

Though the recent GSI controls system is already designed as a decentralized distributed system, it suffers from the dependency on in-house communication and network protocol, current hardware and software.

The front ends use Motorola 68000 single board computers with VME bus running on pSOS, while the applications for accelerator operation are running on Alpha workstations running OpenVMS. Most of the applications are written in Fortran or Pascal, some in Nodal.

The GSI controls system models the equipment of the accelerator as independent devices with "properties". Calling a property means supplying the device with new data, like setting the reference current, reading data from the device, or initiate an action like reset. They are implemented as functions on the front end computers, called user service routines (USR), which handle all activities which are specific for the particular type of devices. Properties are grouped to "equipment models", one for each type of devices. Although implemented procedurally, the structure reflects well the object oriented paradigm. Equipment models correspond to classes, properties to methods. The equipment of the accelerator is represented in the control system by object-like devices on which properties act.

* k.hoeppner@gsi.de

Front End Level Upgrade

On the front end level, the upgrade to VME boards with PowerPC CPUs running embedded Linux and using CORBA as middleware is in progress [1]. Meanwhile, devices are implemented in an object oriented approach, using a set of generice C++ classes (`AccDevice`, `AccData`) that is independent of CORBA as communication interface.

The USR code for the old equipment models is mostly reused in the new equipment models as methods of the device classes. Thus, a large set of equipment models could be migrated successfully to the new device classes (see Fig. 1).

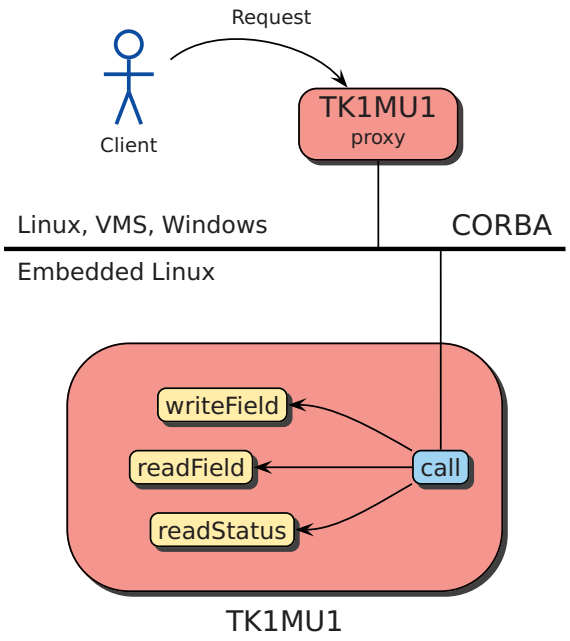


Figure 1: Access to new devices using CORBA as middleware. Old USR code is reused as methods of device classes.

Upgrade of Operating Level

Since most of the applications for accelerator operations are written in Fortran or Pascal, heavily using VMS system libraries, many efforts have to be made for an upgrade of the operating level.

Currently, the usage of commercially available VMS emulation libraries is under evaluation, hopefully providing the opportunity to migrate legacy applications to Linux. On the other hand, for new devices being used as part of the current accelerator, they have to be accessible from the

current applications running on OpenVMS via the in-house communication protocol called Userface. A new Userface server under Linux is under development [2]. It will act both as a traditional Userface server providing access to old devices and translate Userface requests to CORBA requests for new devices on the fly.

In parallel, the development of client interfaces independent from the old VMS applications is in progress. For example we created a Python API in addition to the C++ client interface that might be used as an successor for Nodal scripting in the future [3].

JAVA API TO ACCESS DEVICES VIA CORBA

First Approach

In parallel to the new Python client interface, we wanted to get a Java API to access new devices via CORBA. It was implemented by Cosylab on top of the CORBA to Java bindings produced by the Java IDL compiler. This is meant as a first step for future Java applications for accelerator operation. Though currently dependent on CORBA as middleware, it is implemented as a semi-abstract layer for device access. So extending the Java API to support other communication protocols should be possible quite easily.

The design of the IDL interface for CORBA access was done with having the coexistence of old and new devices in mind. Thus, a narrow interface was chosen for compatibility reasons. This lead to strong consequences for the implementation of the Java API. Since it provides Java Beans supporting a wide interface for device access, it has to map the wide interface to the narrow CORBA interface internally.

Problems During Implementation

Some of the problems that occurred and were solved during implementation of the Java API were owed to the special kind of operation at GSI.

GSI is operated in a pulse-to-pulse mode switching between different ion sources and several experiments in parallel (see Fig 2). This special multiplex mode was mapped

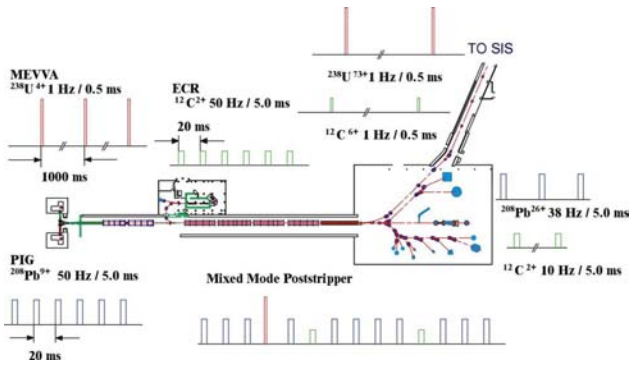


Figure 2: Pulse-to-pulse operation mode at GSI.

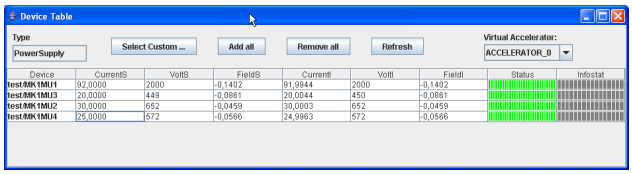


Figure 3: Table with the states.

to a concept of so called *virtual accelerators* in the current GSI controls system. The Java API has to take this concept into account, i. e. the Java API provides the possibility to set reference values or measure actual values for current, field etc. for a given virtual accelerator.

Another problem occurred from the fact, that in the current GSI equipment models the device properties don't contain additional information. No description of the properties like minimum and maximum value, alarm or warning levels is available from the device itself, since they are currently provided by the central operating database on OpenVMS. Since the new C++ device classes on front end level reuse the existing code from equipment models, this information is inaccessible to the Java API for device access.

We plan to extend the new equipment models and the CORBA IDL definition. Then, device servers will provide additional information about the properties that can be used by Java clients e. g. for choosing ranges of axes or for triggering alarm messages.

APPLICATIONS USING COSYBEANS

Cosylab implemented some applications using CosyBeans on top of the new Java client API. These applications give a first look on the potential of Java GUI applications

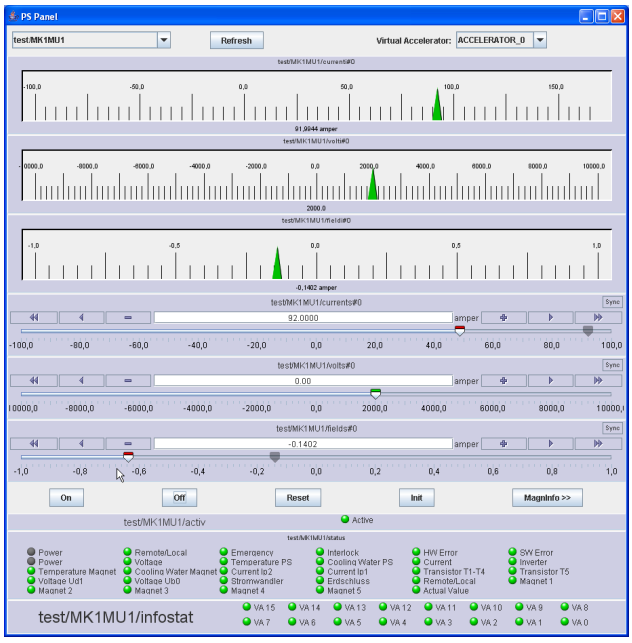


Figure 4: Panel for a power supply.

for accelerator operation at GSI.

Some effort had to be made to reproduce the typical style of GUI applications at GSI that normally use *one* widget switchable to *many* devices. Additionally, the CosyBeans application support the concept of virtual accelerators used at GSI. Some screen shots are shown in figures 3 and 4.

In the current state, due to the "dumb" properties of the GSI equipment models, some manual configuration of the applications is necessary, e. g. the display range of gaugers has to be set manually. As said before, it is planned that devices will provide additional information about the properties in the future, so in upcoming releases the configuration of CosyBeans widgets will be done automatically.

REFERENCES

[1] K. Höppner, L. Hechler, P. Kainberger, U. Krause, "Embedded Linux and CORBA in GSI Controls System", PCaPAC 2005, March 2005, Hayama, Japan, http://conference.kek.jp/PCaPAC2005/.

[2] G. Schwarz, L. Hechler, P. Kainberger, U. Krause, K. Höppner, G. Fröhlich, V.R.W. Schaa, to be published.

[3] L. Hechler, K. Höppner, P. Kainberger, U. Krause, G. Schwarz, "Replacement of Outdated VME Boards as Starting Point for Controls System Modernization", ICALEPCS 2005, September 2005, Geneva, Switzerland, http://icalepcs2005.web.cern.ch/icalepcs2005/.

PC-BASED INNOVATIONS IN THE VEPP-4 OBSOLETE CONTROL SYSTEM

V. Kaplin, S. Karnaev, O. Meshkov, I. Morozov, O. Plotnikova, V. Smaluk, A. Zhuravlev,
BINP, Novosibirsk, 630090, Russia.

Abstract

The VEPP-4 control system was designed 20 years ago as CAMAC-based control system [1]. Over this period of time the requirements for the control operations increased significantly. The experiments on high precision measurements of Ψ-meson and τ-lepton require exactly know the energy of beam particles which depends on a lot of accelerator parameters. Due to PC's integration the VEPP-4 control system is fully adequate for the present day tasks to control the VEPP-4 facility. The using of PC's allows us to develop some systems for high precision measurements of the beam parameters and automatic control of the accelerator operations.

INTRODUCTION

The most important factor determining the average energy of the circulating bunches particles in the VEPP-4M collider is the temperature of the dipole magnets [2].

The integral value of the magnetic field in the beam orbit area depends on the magnet dimensions, which depend on a temperature. Thus, for the estimation of the average energy of bunches, it is necessary to measure the temperature of the magnetic elements precisely and to provide thermal stability of the all magnets.

The measuring of the VEPP-4 collider RF cavities temperatures is important also. The RF cavities temperature is stabilised with special system [3] in order to avoid phase oscillations of the beams.

The special automatic system was developed in order to prevent damage of the chamber. The special electronics and program in PC [4] indicate appearance of the phase oscillations and send information to the control program which moves the counter beams apart.

THE STRUCTURE OF THE TEMPERATURE MEASUREMENT SYSTEM

The new VEPP-4M temperature measurement system bases on BINP developed 32 channel temperature controllers using High-Precision Digital Thermometers DS1631 with the resolution 0.0625º C [5].

High-Precision Digital Thermometers DS1631

DS1631 is produced by MAXIM/DALLAS Company. Sensor's principal features are:

- DS1631 provides ±0.5° C accuracy within 0° C up to +70° C range;
- operating temperature range: –55°C to +125° C;
- temperature measurements require no external components;
- output resolution is user-selectable to 9, 10, 11 or 12 bits (12 bits resolution corresponds 0.0625° C);
- wide power supply range (2.7V to 5.5V);
- converts temperature to digital word in 750 ms (max);
- data is read/written through 2–wire serial interface.

32- channel Temperature Controller

32-channel Temperature Controller was developed in BINP (see Fig. 1). Controller's functions are:

- data is read from the temperature sensors every second and is written to the memory of the controller;
- automatic check of the temperature value of each sensor to be inside the specified temperature range;
- switching on the relay interlock if the temperature is out of the specified range.

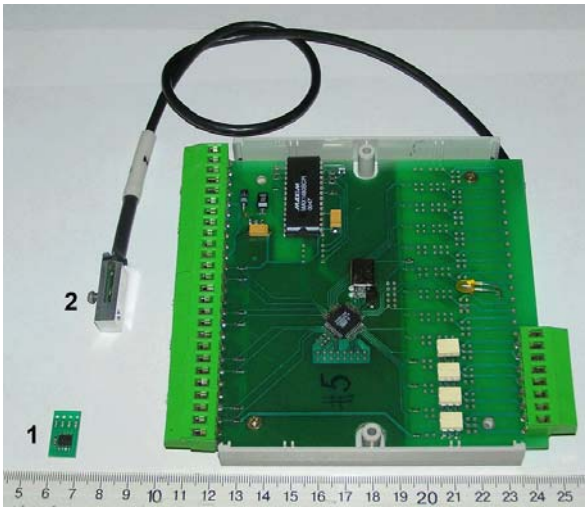


Figure 1: The temperature controller:
1 - sensor board; 2 – sensor inside the protection case.

Controller is connected with sensors through four-wire multi-drop serial lines. It may be connected up to eight sensors in one line of 20 meters length. Temperature values are renewed for the all sensors of each controller and stored to the controller's memory every second automatically.

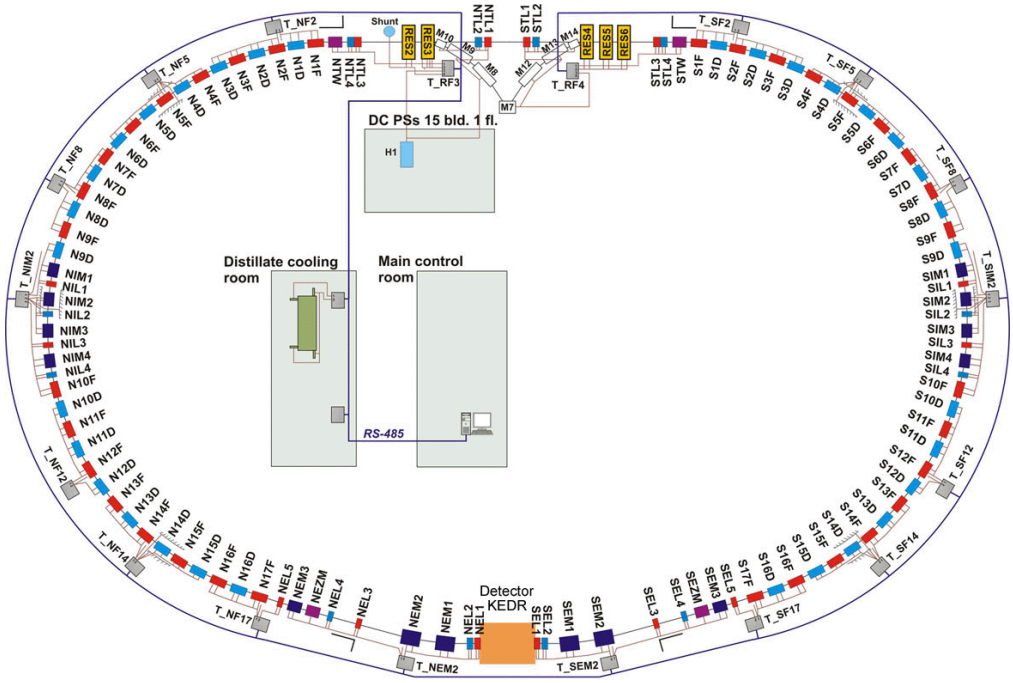


Figure 2: Lay-out of the new VEPP-4M temperature system.

If some of the temperature values exceed specified temperature limits, then specified relay contact is closed automatically. Relay interlock is proposed to be used to prevent the VEPP-4M magnetic elements overheating. There are up to 12 solid-state relays in one controller. PC program reads from and writes to controller the specifications for relay contact closure. Controller closes relay contacts automatically this specification but can't opens its. It shall be open by the control application running in PC.

The temperature measurement system lay-out

It may be connected up to 30 controllers in parallel to one link via serial port in PC. The distance from PC to the last controller is up to 1200 meters. Fig. 2 illustrates the sensors and controllers distribution at the VEPP-4M ring.

RS-485/RS-232 interface and specially developed protocol are used for the connection between the temperature controller and the PC.

The tunnel air and tunnel walls temperatures are measured in several points in order to provide the estimation of the ring geometry. Each magnet is measured in two points: on upper and down parts of the yoke. The magnets temperature values are taking into account in the field behaviour estimation between the energy calibrations. The temperatures of the main bending field power supply and the measuring current shunt are important for the energy estimation.

The temperatures of cooling water are also important for the energy estimation, so the cooling water temperature is measured in several points. The inlet and outlet cooling water temperature is measured for each RF cavity in order to check the temperature stabilisation system.

THE TEMPERATURE SYSTEM SOFTWARE

The program running in PC reads the data from the all controllers and writes the data to PostgreSQL database once per minute. The stored temperature data are used for the particles energy estimations in the high energy experiments.

The graphic interface provides browsing of the temperature diagrams of the selected sensors over any period of time. The typical temperature diagrams are presented in Fig. 3.

The configuration of the controllers and sensors are stored into PostgreSQL database too. The monitoring program reads the configuration data from the database for renewing periodically. It provides permanent measurement and storing of the temperature data if the measurement system is modified. All the programs run under Linux and use Motif's library.

THE BEAM PHASE OSCILLATION MONITORING SYSTEM

System configuration

For the purpose of monitoring of synchro-betatron resonances and phase oscillations, and studying of beam collective effects the Fast Profile Meter (FPM) based on HAMAMATSU multi-anode photomultiplier tube was designed in BINP [4]. FPM is a part of the VEPP-4M optical diagnostic system [6], and it consist from the 16-channel multi-anode photomultiplier, a fast 12-byte ADC, a controller with internal memory of 4Mb, and 100 Mb Ethernet interface. FPM can record more 100000 profiles of a beam at 16 points. It allows

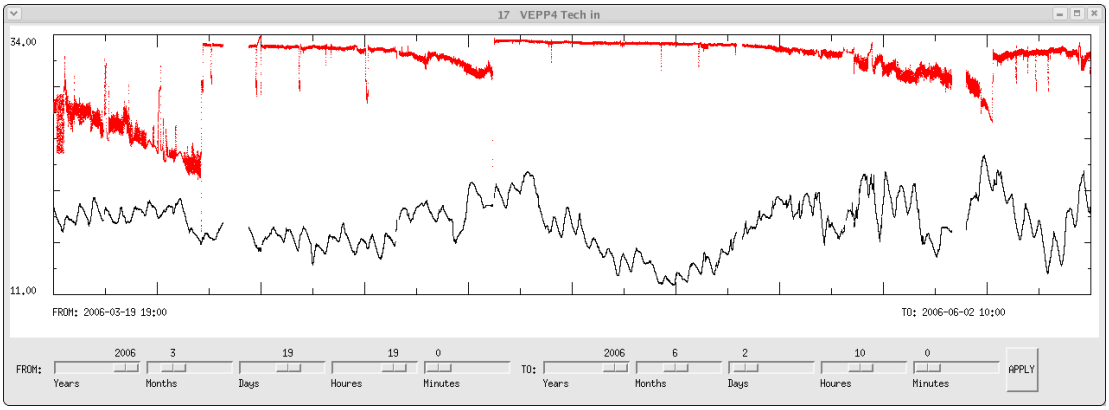


Figure 3: The out circuit cooling water temperatures during two and half months (red line – out-stream temperature, black line – in-stream temperature).

us to analyze the frequency oscillation of a beam in the range from 10 Hz up to 1MHz. The scheme of the measurement system is shown in Fig. 4.

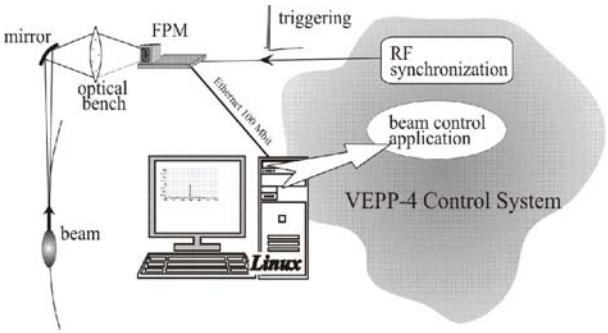


Figure 4: The phase oscillations monitoring system scheme.

System operation

The phase oscillations monitoring system measures the radial position of the beam and executes Fourier analysis of the signals from each of the 16 channels permanently. The maximal value of the Fourier harmonics corresponding to synchrotron oscillations is calculated from the signal coming from the photomultiplier channel which cathode is corresponded to the beam edge in transverse plane. The diagram of the indicated oscillations is shown in Fig. 5.

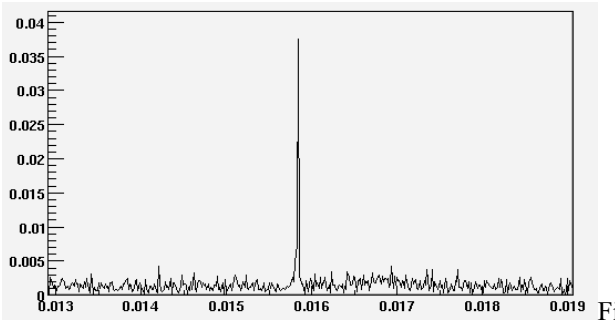


FIgure 5: The phase oscillations harmonics.

The maximal harmonics value is sent to the VEPP-4 control system. When the oscillations appear and the measured harmonic value exceeds the allowable level the beam control application running in Odrenok computer moves the counter beams apart. It prevents the detector drift chamber from parasitic electrical discharging.

CONCLUSION

The using of PC's allows us to improve the possibilities of the VEPP-4 control system.

The new temperature measurement system was developed on the VEPP-4M collider in order to provide the precise and continuous temperature measurements of the all important parameters and parts, which effect on the beams energy. Using the resonant depolarization technique it allows us to estimate permanently the beams energy with the accuracy up to 5 x 10^{-6} (10 keV) [7].

The new features of the beam operations automation allow us to carry out the series of Ψ-meson and τ-lepton mass measurement experiments on the KEDR detector.

REFERENCES

[1] A.Aleshaev, et al., "VEPP-4 Control System", ICALEPCS'95, Chicago, USA.

[2] V.E. Blinov, et al., "Absolute calibration of particle energy at VEPP-4M", Nuclear Instruments and Methods in Physics Research, A, 2002, 494 (1-3), pp. 81-85.

[3] E.G. Miginskaya, et al., "Management in temperature of RF-cavities of VEPP-4M Electron-Positron Facility", talk on PCaPAC'2006.

[4] O.I. Meshkov, et al., "Application of the beam profile monitor for the VEPP-4M tuning", talk on EPAC'2006 ?.

[5] http://www.maxim-ic.com/parts.cfm/p/DS1631

[6] O. I. Meshkov et al., "The upgraded optical diagnostics of the VEPP-4M collider", Proceedings of EPAC 2004, Lucerne, Switzerland, pp. 2739-2741.

[7] V.V.Anashin, et al., "New precision measurements of the J/Ψ and Ψ' meson masses", Physics Letters, 2003, v.B573, p.63-79, hep-ex/0306050.

BUILDING AND DEPLOYING LOOSELY COUPLED CONSOLE APPLICATIONS

Andreas Labudda, Deutsches Elektronen-Synchrotron DESY in der Helmholtz-Gemeinschaft, Hamburg, Germany.

Abstract

The set of computer platforms foreseen for the new accelerator Petra III [1] is much more heterogeneous than that of Petra II [2]. DESY expects to use Petra III clients in several administrative contexts as well. The goal is to build platform independent client applications and deploy them irrespective of the user requesting the application. The resulting strategy should have as little impact on application development as possible and make use of existing technologies where they exist.

In order to remain platform independent JAVA [3] was selected as the development language and Eclipse [4] as the development tool. The applications will be deployed over the boundaries of administrative contexts by anonymous http and Java Web Start [5]. To support the application development a "quick start" wizard application was designed, which collects the application's metadata. This information is used by a customized ANT [6] script which builds, signs, and deploys the console applications. In this paper we describe "quick start" and the four targets of the DESY ANT script.

OVERVIEW

In the process of building a new control system for the new accelerator PETRA III, the control system software for existing accelerators will also be updated. The project plan is tight. To get ready in time, already existing hard- and software components are used for development. For example the developer PCs are supported by the central IT staff just like other office PCs and central file space is used, including central backup. Software packages like Eclipse have been distributed with a DESY common configuration.

The design decisions for support of the accelerator operations are very different from those for the developer environment. For operations the main point of interest is not using existing components, rather the availability of the control systems. Third party services (such as those from central DESY IT) will not be used in accelerator operations if there is no service caching algorithm or if interruptions to the service are not acceptable.

To enable the developers to focus on designing and implementing applications the deployment process has to be transparent. Setting up a new project is like starting any other console application in the control system. An application, the so called "New Project Wizard", is launched via control systems launch mechanism. The "New Project Wizard" generates a new Eclipse project. The generated project complies with all requirements of the "Common Build and Deploy" process. Each console application in the "Common Build and Deploy" can be released by one click without regard for other console applications, administrative boundaries or destination operating system.

BUILD AND DEPLOY ENVIRONMENT

The build and deploy environment consists of several parts. The parts are just used like the DESY IT infrastructure, but installed and configured like the source code versioning system or custom made like the "New Project Wizard"

DESY IT infrastructure

The developer environment is largely integrated into the DESY IT environment.

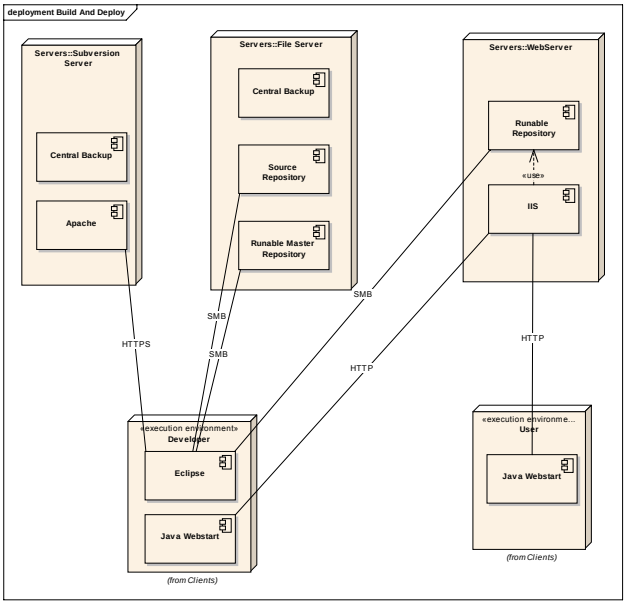


Figure 1: Developer PC as embedded in DESY infrastructure

The DESY IT division offers support for

- System setup for installation and configuration of the operating system.
- Application installation and configuration. Office, Java SDK, Eclipse including some major plug-ins.
- Disk space on file and web server.
- Backup Service.
- Login service and account management.

There is no central source code versioning service available. This service is made locally by DESY-MST.

New Project Wizard

The "New Project Wizard" is a custom java application available from the control system launch service. The

wizard enables a “quick start” to create a new project and integrate it into the build and deploy process. A folder structure is created and files are generated by the wizard. Folders and files contain all information needed by the build and deploy process to make an application available for a console.

Build and Deployment

As at CERN [7] the “Build and Deployment” process is made by ANT. The process consists of two parts. Part one is a “Private Build and Deploy” project script. It is created by the “New Project Wizard” in each new projects folder. The “Private Build and Deploy” script declares four jobs or public ANT targets to be done. Within this file inside the project folder Eclipse is able to display these four targets and use part two of the build and deploy process. The second part is also an ANT script, the “Common Build and Deploy” script. It implements the targets defined by the project’s “Private Build and Deploy” script.

Versioning

Subversion [8] was selected as code versioning system. It is commonly used via Eclipse plug-in, but other clients are also available. Source code is going to be committed on every release of the project, irrespective of the kind of release. If the project is released for production a new tag is created.

Launch

The launch mechanism depends on “Java Web Start”. Java Web Start consists of server and client parts. The server has to support Hypertext Transport Protocol (HTTP) and make available the Java Network Launching Protocol (JNLP) [9] files and the required Java Archive files.

The client computer has to provide a JNLP client. In this environment the Sun Microsystems, Inc “Web Start Client” is used. The client caches downloaded files if a newer file exists on the web server. To start and run already downloaded code the web server is not needed.

NEW PROJECT WIZARD

The New Project Wizard enables a developer to include a new Project into the build and deploy process in an easy way. The minimum information the wizard needs is

- Selection of the accelerator to which the new application is dedicated.
- Selection of an available subsystem or the name of a new subsystem.
- The name of the project.

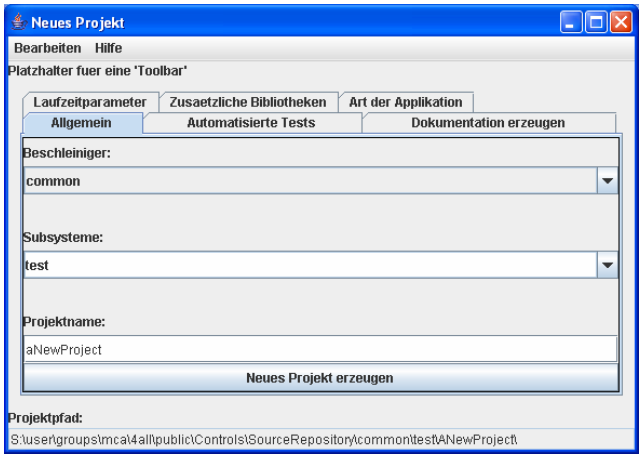


Figure 2: Screenshot of the “New Project Wizard”

After collecting this information the wizard creates a new Eclipse project with the following properties:

- A source folder with a default package structure.
- Basic source files
- Subfolders for compiled sources, java doc files, test classes, deployable files.
- Files to start and control the build and deploy process
- Eclipse project files

BUILD AND DEPLOY WITH ANT

ANT is integrated into Eclipse. ANT targets are easily startable by using the Eclipse ANT view. In this environment four public targets are defined.

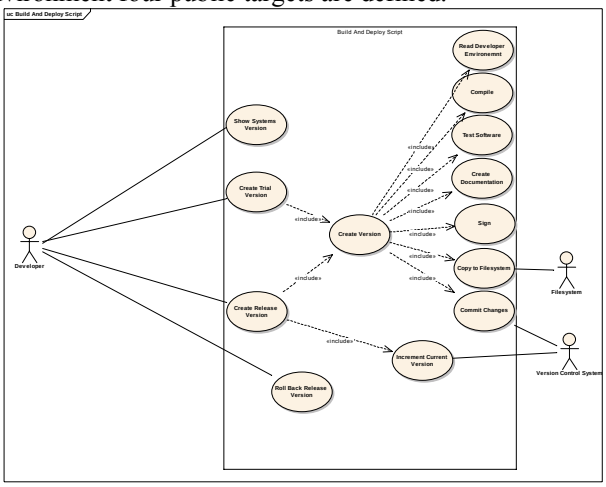


Figure 3: Public ANT targets and used private targets

Version Information

The target Version Information shows the version of the build process. Because the implementation of the build process will change, the user has to have an idea of the state of the implementation.

Trial Release

Release for evaluation enables the developer to offer a new version of an application without interfering with accelerator operation. This target consists of the following private targets

- Compile all existing sources.
- Execute JUnit [10] tests if available.
- Create jar Packages including sources, resources and binaries.
- Commit changes into versioning system.
- Sign the package.
- Create JavaDoc [11] files
- Deploy to web servers

Production Release

Release for production releases the application for accelerator operations. The next time the application or lib is loaded; this new version will be downloaded from the web server. Release for production extends the release for trial to enable version numbers. Each release for production increments the version number of the project and creates a new tag in the subversion repository.

Rollback

Rollback is to be used to make the previous released version of the project available for accelerator operation.

FURTHER DEVELOPMENT

One of the design goals of the build and deploy process is the ability to change implementations in case of increasing knowledge. The basic components like Eclipse, ANT and Web Start are hardly subject of change. It is also not easy to change the number and names of public ANT targets in a large number of existing independent projects.

Because the ANT script is split into "Private Build and Deploy" and "Common Build and Deploy" sections it is easy to change the implementation of the common targets. Because the "New project Wizard" is launched via Web start it can be updated frequently.

ANT script improvements

The first version of the ANT script was built to use within Eclipse 3.1. The ANT tasks shipped with Eclipse 3.1 does not satisfy all requirements so some ANT tasks had to be downloaded from the open source community. Now Eclipse 3.2 is available. In contrast to the previous version of Eclipse more ANT tasks are included inside the current version of Eclipse. It must be checked whether some of the separately downloaded ANT task are obsolete or could be replaced by smarter ANT tasks shipped with Eclipse 3.2.

Wizard integration

To increase development performance and maintain-ability a device server wizard [12] was created. Up to now they are separate projects. In future they should be integrated with each other.

REFERENCES

[1] R. Bacher, P. Duval, S. Herb, U. Lauströer, R. Schmitz, W. Schütte, "PETRA III TDR Chapter 3.12 General Control System", http://adweb.desy.de/mst/PETRA_III_Kontrollsystem/TDR_22_10_03.pdf

[2] Rüdiger Schmitz "A control system for the DESY accelerator chains", http://adweb.desy.de/mst/Mst_content/Vorbeschleuniger_control_systems_1999.pdf

[3] Sun Microsystems, Inc, Java, http://java.sun.com/

[4] The Eclipse Foundation, "Eclipse - an open development platform", http://www.eclipse.org/

[5] Sun Microsystems, Inc, "Java TM Web Start", http://java.sun.com/j2se/1.5.0/docs/guide/javaws

[6] The Apache ANT project, ANT, http://ant.apache.org/

[7] Grzegorz Kruk, CERN, AB/CO/AP , "Development Process of Accelerator Controls Software" http://icalepcs2005.web.cern.ch/icalepcs2005/Presentations/14oct_Friday/FR2/FR2_5-6O.ppt, ICALEPS 2005

[8] Tigris.org, Subversion, http://subversion.tigris.org/

[9] Java Community Process, "Java Network Launching Protocol and API", http://jcp.org/en/jsr/detail?id=56

[10] Junit.org, Junit, http://www.junit.org

[11] Sun Microsystems, Inc Javadoc, http://java.sun.com/j2se/javadoc/

[12] Josef Wilgen, DESY, MST, "A Device Server Generator for Control Systems", See this proceedings.

RF SYSTEM HIGH POWER AMPLIFIER SOFTWARE CONVERSION AT JEFFERSON LAB

George Lahti, H. Dong, T. Seeberger
Jefferson Lab, Newport News, VA 23606, U.S.A.

Abstract

Jefferson Lab is in the process of converting the RF system from analog RF modules and non-smart high power amplifiers (HPAs) to digital RF modules and smart HPAs. The present analog RF module controls both the RF signal and the non-smart HPA hardware. The new digital RF module will only control the RF signal, so the new HPA must include embedded software. This paper will describe the conversion from a software perspective, including the initial testing, the intermediate mixed system of old and new units, and finally the totally new RF system.

HISTORY

The original HPA had no firmware. The digital/ADC/DAC control was handled via the analog RF Module and the EPICS IOC, or by the TACL Controller in the early days. The IOC handled only the on/off digital controls and, a few (3 or 4) analog monitored signals, and one interlock. The RF Module handled all other signals, both set points and measured values, as well as all other HPA interlocks.

The present analog RF Module has two purposes. The primary purpose is to generate stable RF that is amplified by the HPA and sent to the RF cavity. The secondary purpose is to do most of the software control of the HPA. It has an embedded Intel 80186 processor using C language software with no operating system, and was developed in 1989. [1]

PRESENT PROTOTYPE GOAL

The goal for this part of the project, for both hardware and software, is to cause as little disruption and change to the whole RF system as possible, while testing the new HPA prototype. All, or most, of the existing signal names will remain the same, so that the screens remain the same, except for any new functionality.

The HPA software that is in the RF Module will remain unchanged. However, the interlock trip point software limits will be set out of bounds, so that we will never get a fault trip. The cable to the RF Module DACs will be disconnected, and will go to the new hardware/firmware HPA module. The cables to the old Module ADCs will be disconnected also, and a "shorting" cable plug will be used so that all the ADCs voltages are zero, ensuring that there are no false interlock trips from the RF Module.

The Epics IOC Idle/Filament/High_Voltage/Rf_On state machine will be modified, so that the Filament and High_Voltage commands go to the new HPA module. Since the present RF Module will not be altered, "dummy" Filament and High_Voltage commands will also be sent to the module, so that it can stay in sync with the new HPA module and be able to go into the Rf_On state (i.e. the present RF Module requires to be in the High_Voltage state before it can go into the Rf_On state).

The original requirements are still to be used, but with additions of new functionality that is provided by the new HPA module. Also, as much as possible, we will reuse the present RF module code that deals with the HPA.

Since the new HPA will have firmware, a new feature must be added to the Epics state machine to keep it in sync with the HPA hardware, when the user puts this hardware into local mode. Any changes that the user makes in local mode will have to be made to the state of the Epics state machine, so that when the user goes back to remote mode, there will be no bump. This bumpless transition will also have to be honored for an Epics IOC reboot, just in case the HPA was put into local mode and changes were made while the IOC was turned off. When the IOC is rebooted, it will read the state of the HPA hardware and set the state of the IOC state machine to match. This feature will allow the IOC to be rebooted without turning off filaments or high voltage, which will save time by not waiting for the filaments to warm up. Since the new HPA module will handle all machine protection interlocks for the HPA, there will be no potential harm to the hardware when the IOC is off.

To roll back from the prototype, we would do the following:
(1) Unplug the new cables and plug in the old cables.
(2) Roll back the software in the IOC via a reboot.
(3) Download the operational trip interlock levels to the present RF modules. No reboot is required.

The prototype HPA module will communicate with the Epics IOC via the CANS Serial Bus.

Three major items that are presently handled by the Epics IOC or external UNIX programs will be moved to the new HPA module.
(1) Miram filament voltage optimization process,
(2) Interlocking of run away klystron mod anode current (KMAI) to protect the klystron hardware from damage, and
(3) AC power line usage economization by varying the mod anode voltage so that the klystron is not running at full beam current when not needed for the present operating point.

Three new major items will be added to the HPA module, which the present system doesn't have: (1)

statistics, (2) full diagnostics mode, and (3) improved fault recording.

The new HPA will keep statistics, such as the hardware serial numbers, firmware version number, and on and off times for both filaments and high voltage. These will be stored both in the HPA module and in some external database.

The new HPA will have improved diagnostics capability and fault recording. The firmware and hardware will be able to perform self tests, go into simulation mode (with the required safety and hardware protection interlocks), and run various diagnostic monitoring. There will be high speed internal ring buffers for all key hardware signals. When a fault (i.e. trip off) occurs, the buffers will be turned off and an analysis done to see what the first fault was and possibly what event caused it. These buffers can be sent to Epics so that we can view these waveforms on the screen.

FUTURE INTEGRATED GOAL

The final transition step would be replacing the present analog RF modules with the new digital RF modules [2]. Then replace the prototype HPA module with the final HPA module. The final HPA module will contain its own built-in Epics IOC, and communicate over the Ethernet. For any interlock communication between the RF module and HPA module, there will be a direct hardware line, not via Epics Channel Access. The goal is to keep the RF running even if communication is lost with the main Epics IOCs.

Since the digital RF modules only handle RF, it only has two states: Rf_Off and Rf_On. The HPA state machine will not have to send dummy states commands that were needed for the present analog RF modules. This state machine separation between the RF and HPA will allow a cleaner user interface on the screens and better organized maintenance and diagnostics.

By having both the digital RF Modules and HPA module connected directly to the Ethernet, they can be more self contained and independent from the rest of the global Epics IOCs. That will improve uptime, since these global IOCs could be rebooted without bringing the RF system down, so not having to wait for the filaments to heat up, nor having to turn off the accelerator beam in some cases.

REFERENCES

[1] George Lahti, C. West, I. Ashkenazi, “Embedded Software for the CEBAF RF Control Module,” Poster compiled for Particle Accelerator Conference, San Francisco, Calif., May 6-9, 1991. (PAC 91), lahti@jlab.org

[2] C. Hovater el al, “High Gradient Operation with the CEBAF Upgrade RF Control System,” 2006 LINAC Conference, Knoxville TN, August 2006

ADVANTAGES OF THE PROGRAM-BASED LOGBOOK SUBMISSION GUI AT JEFFERSON LAB*

T. McGuckin, Jefferson Lab, Newport News, VA 23606 U.S.A.

Abstract

DTlite is a Tcl/Tk script that is used as the primary interface for making entries into Jefferson Lab's electronic logbooks. DTlite was originally written and implemented by a user to simplify submission of entries into Jefferson Lab's electronic logbook, but has subsequently been maintained and developed by the controls software group. The use of a separate, script-based tool for logbook submissions (as opposed to a web-based submission tool bundled with the logbook database/interface) provides many advantages to the users, as well as creating many challenges to the programmers and maintainers of the electronic logbook system. The paper describes the advantages and challenges of this design model and how they have affected the development lifecycle of the electronic logbook system.

INTRODUCTION and HISTORY

The "Downtime Log lite" (DTlite) was originally written by an operator in the Jefferson Lab control room (MCC) to assist in making reports of accelerator downtime to the html-based electronic logbook. It was a very straight-forward tool, allowing input of times, a location, keywords, a description of the problem and capturing of no more than two images.

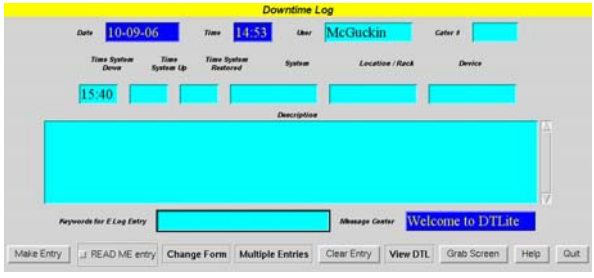


Fig. 1: v1.0 of DTlite

One of the key features of this earliest version of DTlite was the ability to grab screen shots from the EPICS computing environment. This allowed operators to capture key information and diagnostic screens for system experts to review later.

This was accomplished by having DTlite (itself a Tcl-script) execute a shell command to launch a screen capture tool (xgrabsc) and save the picture to a file and then inserting an html-reference to the file into the logbook entry.

* This work was supported by DOE contract DE-ACo5-84ER40150 Modification No. M175, under which the Jefferson Science Associates (JAS) operated the Thomas Jefferson National Accelerator Facility.

Other scripts were also available including a table generation and data gathering script. Further incremental changes were made over time to the script to provide additional features.

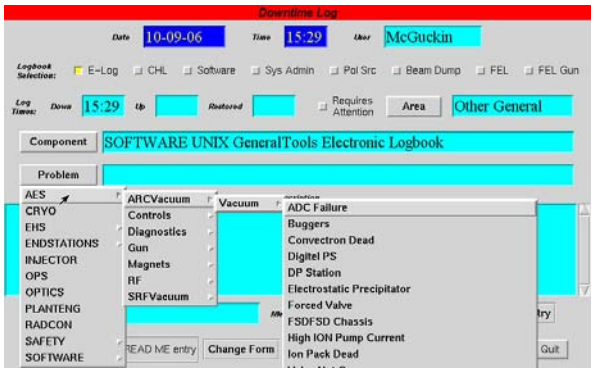


Fig. 2: v3.0 of DTlite demonstrating several upgraded features

- Logbook Selection buttons across top allow entries to multiple logbooks
- Static Area/Component/Problem menus provide clearly defined guidelines for correct information
- Color coordination to provide easy visual identification of log type (red = downtime)

In January of 2002 a major upgrade of the electronic logbook was done, converting it from a series of linked html entries to a full database structure.

Concurrent with this, version 4.0 of DTlite was upgraded to be compatible with the new database logbook structure. Version 4.0 also included dynamic menu structures that were generated by querying the database when a downtime entry was started.

Several other features were also added shortly after this:

- The old limit of two screen captures was removed, allowing any number of pictures to be attached to an entry
- Time checking to confirm that correctly formatted and ordered values are provided
- Enhance email functionality (to email entries to system experts)
- Linking of entries so that similar entries contain links to each other in the database

The next major change occurred in August of 2004 when the Jefferson Lab control room was renovated and a new video display wall was added.

This change required several feature upgrades to DTlite. The most important of which was the ability to "share" open log entries such that one operator could start an entry, then it could be stored (referred to as "walled" because the entries are pushed to the display wall) and

then fetched by another operator to be updated/submitted. This was especially important for entries that would span multiple shifts and therefore had to be stored somewhere other than on a user's local account.

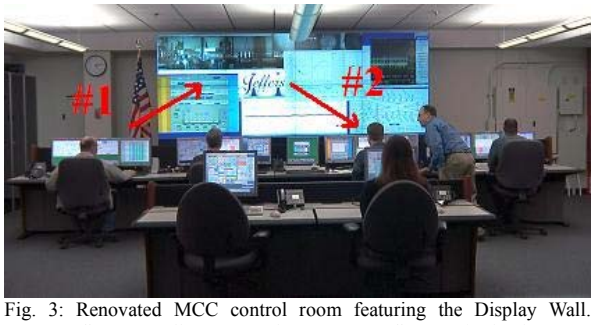


Fig. 3: Renovated MCC control room featuring the Display Wall. Operator #1 can "wall" an entry; later Operator #2 can "fetch" the entry, modify it, "re-wall" it or submit the entry to the logbook

Along with this upgrade several tools were added to retrieve "lost" entries (due to machine crashes or accidental user action).

One of the most important features that were developed in this same time period was the addition of system-expert-provided guidance to the problem reporting system of DTlite. The purpose of this system is to provide a first-level of support for operators for common problems encountered in the accelerator.

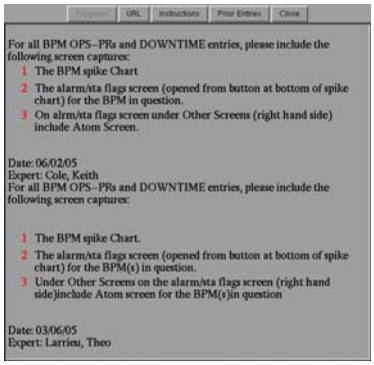


Fig. 4: For a given component selection for a problem report, experts can make guidance available, either on what information to gather, or on how to troubleshoot the problem

Shortly after this (Sept. 2004) the DTlite program was added to the cvs-based Control System User Environment (CSUE). Incorporating DTlite into CSUE allowed for greater versioning control and, more importantly, tracking of changes to the script.

Several other features have been added over the last two years to increase DTlite's functionality;

- the ability to attach files (allowing for attachments other than pictures)
- behind-the-scenes work to compartmentalize different aspects of the code (increased stability)
- the ability to include a caption with each picture to further enhance information capture
- expanded guidance feature for problem reports to include guidance/entries from previous, similar problem reports
- replacement of HP's screencapture tool with a cross-platform xsnap utility, allowing DTlite to be run natively on OS's other than HP-UX

ADVANTAGES

Using an external program (as opposed to an html/java-based program bundled with the web browser used to view the electronic logbook) has provided many advantages to the Jefferson Lab accelerator environment. Likewise the utility of that program has expanded greatly over its lifetime, which has been largely possible because of the script-based nature of the program.

Script-based – There is a minimum of overhead in running the script. As opposed to having to launch a web browser (for an html or java-based utility) or some other wrapper utility, all that DTlite requires is that Tcl/Tk be built/installed on the computer. Likewise launching the application is simply an xterm command or a wrapper script launched through some method.

This also allows DTlite to use the tkispell spell-checking package built for Tcl/Tk with a minimum of effort or extra work. This allows for a more professional looking logbook overall.

Screen grabs – One of the biggest advantages of the DTlite script is the ability to launch a screen grab utility to capture screen shots and submit them to the logbook. This has proven to be an invaluable tool for capturing accelerator information (such as the machine state and any error signals) into the logbook. This greatly enhances the ability of system experts to investigate (and ultimately solve) problems by examining various screen shots taken at the time of the problem.

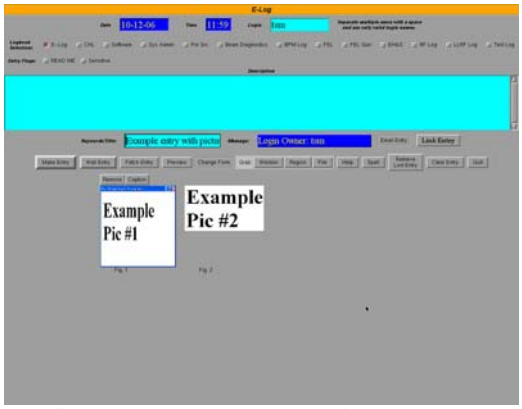


Fig. 5: DTlite screenshot showing two example picture grabs that could be submitted to the logbook to record machine state information

Script launching – Because of the script-based nature of DTlite, it is very easy to launch any other script from within the program. All that is required is a wrapper command and an additional entry in the "Helpers" menu. Likewise DTlite can be launched by another script with values for certain fields (Area, Keywords, Body) filled in.

This allows data to be gathered by a script and then input into a DTlite launched by the script that the user can then add additional information to, or simply submit.

Writing these scripts is also very straightforward, and many users and operations staff have written helpers for DTlite that have later been incorporated officially into the production version. The ease of programming either in a shell language or Tcl/Tk (both of which DTlite can execute from the command-line) contribute to the utility of the script.

Sharing entries – It is very easy for Operations staff to share entries or move them to be stored on the Display Wall using utilities written for DTlite. This functionality is critical to machine operations both from an information sharing perspective, and also because entries can often span multiple shifts, requiring a new operator/crew to have a method for accessing on-going entries.

Caching of information – Because DTlite is run locally (as opposed to being from an app located on a website that has to be connected to) entries can be created and submitted even if the database or network is down. The entries are then stored locally, and when the network/database is available again, a cronjob runs that flushes all the back entries and inserts them into the database.

Likewise, because DTlite is running locally and is not part of a (stateless) webpage app, it is a simple matter for on-going entries to be temporarily saved (in a secure location on the system). This allows for recovery of lost entries due to system crashes or user error.

CHALLENGES

Along with the advantages of maintaining the script-based DTlite program there are, of course, several challenges that are created by this model.

Cross-platform script availability - The first challenge that comes up is that not everyone can run the script. Windows machines don't generally have Tcl/Tk installed, and so can't run DTlite. Likewise there has to be some facility for making entries from locations "off-site" from the lab (for on-call assistance from home, for example). Because of this a more widely accessible web-based log entry program has to be maintained. While this program is not required to have all the features of DTlite (no screen captures, for example), there must be some parallel development and maintenance between the two programs.

Efficiency of script functions - Another challenge to overcome is based in the scripting-language of DTlite. Some functions performed by the script can take a noticeably long length of time. For example, when the Area, Problem and Component trees are built for Problem Reports (see Fig. 2); the tree structure must be parsed and built. This can take a noticeable length of time (5-10 seconds) and can slow down work in the control room if they are trying to gather information.

As these trees become more refined and are expanded, the time issue becomes more important to address. At some point it could become the case that the trees will be so large that the current parsing method will no longer be viable. New techniques (such as storing a "live" tree list locally, rather than re-generating it each time a new DTlite is launched) are under investigation to address these processing overhead issues.

Keeping users updated - DTlite is a heavily used tool and it can often be a challenge to keep all users up-to-date with changes to the script (both changes to core features and newly added features). This has resulted in cases of new features being reported as errors (for example, tightening up security on who can make entries can result in users reporting that they can no longer make entries, because they have not yet been authorized, or don't know the proper format in which to provide their login), or simply not being utilized because users have not realized they were available.

Dissemination of information across groups in a facility is always a challenge for any organization. Various techniques (email, update reports, logbook entries, and face-to-face meetings) have been used and new methods are constantly being explored.

Encouraging the of provided tools - Just as dissemination of information across groups can be a challenge, buy-in by system experts from multiple groups has also been a challenge. Features like Guidance for common problems, Problem Report tree structures for Area/Components/Problems, and Helper applications are only as effective as the system experts that maintain the data for them.

Several techniques have been used to encourage system experts to make use of the tools available. The most obvious technique is to demonstrate how providing guidance for the control room operators can decrease the amount of time that system experts have to spend troubleshooting common (and often easily fixable) problems and minimize how often they are called. Likewise having accurate Problem/Component trees for the Problem Reporting increases the accuracy of the PR and cuts down on the amount of "back-tracking" system experts have to perform to understand what happened during a problem report.

CONCLUSION

Maintaining a stand-alone Tcl application for making logbook entries has created its fair list of challenges. Many of these resulted because the project was begun as a side project and didn't receive a structured development life-cycle for many years. But as that life-cycle has been applied and new features have been added, with existing features being upgraded and improved, DTlite has continued to be an invaluable tool for information collection and logging in the accelerator environment.

MANAGEMENT IN TEMPERATURE OF RF- CAVITIES OF VEPP-4M ELECTRON-POSITRON FACILITY

E.G. Miginskaya, I.I. Morozov, V.M. Tsukanov, A.A. Volkov, BINP, Novosibirsk, Russia.

Abstract

Temperature variation of RF-cavities leads to a change of their geometrical sizes that provides undesirable cavity modes and to excitation of phase oscillations. It leads to decrease in luminosity and a beam life time.

Flowing water heaters with stabilization of temperature have been established for elimination of this disadvantage. Temperature probes LM335 were used with a sensitivity of 10 мV per degree centigrade. The power part is made on the controllable switches CPV240. The analysis of temperature of input and output temperatures of water is carried out by microcontroller ADAM connected to a computer by means of interface RS-485.

The temperature variation have been reduced from 5 to 0.2 degrees centigrade. That has led to decrease in probability of occurrence of parasitical phase oscillations more than in 100 times.

INTRODUCTION

VEPP-4M is the collider with the high energy beams of electrons and positrons [1]. The operating mode with two bunches of electrons and two bunches of positrons has been realized at VEPP-4M collider. Probability of occurrence of phase fluctuations of bunches was increased with the bunches number. Principal reason of occurrence of phase fluctuations it re-tuning RF cavities due to change of the geometrical sizes because of change of their temperature. Life time of a beam was decreased with increase of amplitude of phase fluctuations. Transversal beam size was increased too.

The method measurement of a level of phase fluctuations was described in paper (2). The precision measuring of temperature was described in paper (3, 4). The typical levels of phase fluctuations and luminosity are presented in Fig. 1 in case of absence of stabilization of RF cavity water cooling temperature. The temperature variation of water exceeded 2^0 C, that leads to increase in a level of phase fluctuations and to reduction of luminosity. Simultaneously big losses of particles are dangerous for the drift chamber of the detector the KEDR.

THE TEMPERATURE STABILIZATION SYSTEM

For stabilization of temperature of resonators the flowing heaters have been developed, allowing to keep the set temperature with accuracy of 0.05-0.1 degrees centigrade. Water flow rate is about 10 l/min.

The temperature of water is measured on an input and on an output of a heater by means of probes, and processor ADAM operates switches CPV240 to set required power of heating. The sensitivity of LM335 probes is about 10 mV per degree.

In figure 1 a photograph of a heater is shown.

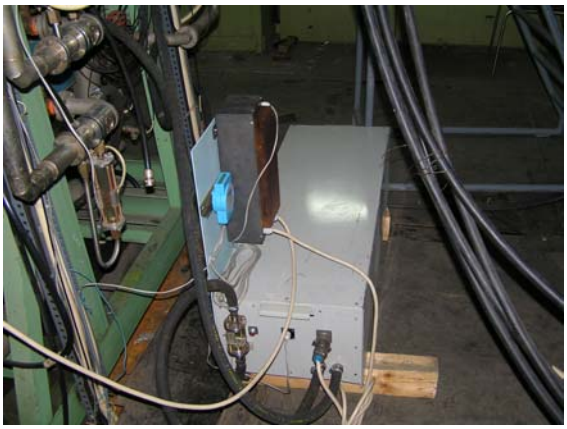


Figure 1: The photograph of heater.

In figure 2 a photograph of ADAM controller and switches CPV240 is shown.

Figure 2: ADAM controller and switches CPV240

In figure 3 the circuit of a heater is shown.

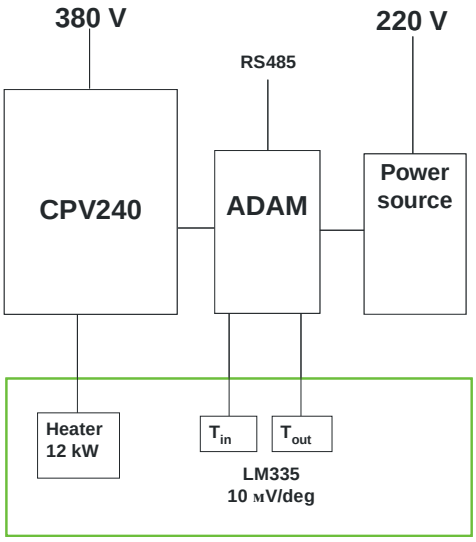


Figure 3:The circuit of a heater

For connection ADAM with control room of VEPP-4M is used interface RS-485. The temperature variation have been reduced from 5 to 0.2 degrees centigrade. That has led to decrease in probability of occurrence of parasitical phase oscillations more than in 100 times

REFERENCES

[1] E.Levichev, A.Bogomyagkov, V.Kiselev, O.Meshkov, N.Muchnoi, A.Naumenkov, S.Nikitin, D.Shatilov, E.Simonov, A.Skrinsky, V.Smaliuk, G.Tumaikin . VEPP-4M Operation at Low Energy The 3rd Asian Particle Accelerator conference, Hotel Hyundai, Gyeongju, Korea, March 22-26, 2004. THM-204.

[2] O. I. Meshkov, A. V. Bogomyagkov, F. Gurko, A. N. Zhuravlev, P. V. Zubarev, V. A. Kiselev, N. Yu. Muchnoi, A. N. Selivanov, A. D. Khilchenko. Application of the beam profile monitor for VEPP-4M tuning . 7th European Workshop on Diagnostics & Instrumentation for Particle Accelerators (DIPAC'05), June 6 – 8, 2005, Lyon, France, POM008.

[3] M. Gluhovchenko, V. Kaplin, A. Kvashnin, I. Morozov. The precision control of temperature VEPP-4M accelerator facility. The 5th International Workshop on Personal Computers and Particle Accelerator Controls. 22 - 25 March, 2005. Hayama, Japan.

[4] V.Kaplin, S.Karnaev, I.Morozov, O.Plotnikova. The precision measuring temperature system of the electron-positron collider VEPP-4M. RUPAC06, September 10-15, Novosibirsk, Russia.

SNS IOCS USE OF RELATIONAL DATABASE TO SUPPLY CONFIGURATION FILE

J. David Purcell, ORNL, Oak Ridge, TN 37830, USA
Wim Blokland, ORNL, Oak Ridge, TN 37830, USA
Andrei Liyu, ORNL, Oak Ridge, TN 37830, USA
Jeff Patton, ORNL, Oak Ridge, TN 37830, USA
Tom Pelaia, ORNL, Oak Ridge, TN 37830, USA
Alexander Zhukov, ORNL, Oak Ridge, TN 37830, USA

Abstract

The Spallation Neutron Source (SNS) Project's Controls group has implemented the use of our relational database as the source of the IOC configuration files. There are almost 500 IOCs deployed at SNS with several variations of operating system and configuration file formats. Until recently, the configuration of these IOCs was left to the individual IOC engineers. Now, new database structures have been created to capture the data contained within the configuration files. New interface tools allow IOC engineers to manipulate their configuration data within the database. After manipulation of the data, the IOCs are triggered to load the new configuration. This paper describes the current experience the steps taken to get it implemented, and plans for the future.

1 OVERVIEW OF SNS RELATIONAL DATABASE (RDB)

The Spallation Neutron Source (SNS) has deployed an ORACLE-based RDB. The RDB was developed to support many different aspects of the SNS project. This includes but is not limited to data structures that support project administration, equipment installation, SNS operations, project documentation and the SNS control system. Because the RDB spans many areas of the SNS project, it has become the central storage area for a vast amount of data and it is considered the main source for information and support data.

Those working with the control system have tried to take advantage of this. The control system "area" within the SNS RDB is the most developed. Many different types of data have been captured. The RDB contains beam line equipment support data, networking data, installation data, calibration data, machine setup data, machine protection system (MPS) data and input output controller (IOC) data. Using the power of the RDB, we can provide a data summary pertinent to anything related to the control system. An example of this would be that we can show what process variables are related to a particular shipment of IOC processors. We can also enforce hardware relationships. We can show that certain connectors have to be used with a certain cable or that a piece of equipment has no open ports. An example that is specific to our initial use of the RDB and the configuration of an IOC is that we can change a channel used by a beam line detector and push that change, through related hardware connections, to the MPS configuration that is used.

2 THE RDB REAL-TIME SUPPORT OF DISTRIBUTED CONTROL SYSTEM (DCS)

The distributed control system (DCS) employs objects that require support data to operate. Because the RDB holds all of the data required (or should), using the RDB to supply data is a natural choice. It can become the single source of information and data used by the DCS (Figure 1). Advantages of one information source include the ability to synchronize the different objects of the DCS, including IOCs/Servers/FECs, OPI/Clients, Alarm server, and Error server. The RDB also increases the ability to control the configuration of the distributed objects and allows tools to be built that can manipulate the RDB contents.

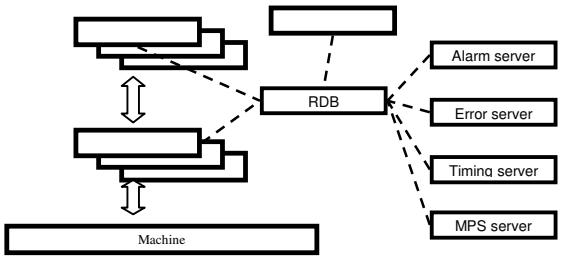


Figure 1 Traditional use of RDB with distributed control system

Traditional communication between the RDB and a specific DCS object can be relatively easy, but communication becomes extremely difficult when looking at the DCS as a whole. The DCS objects are based on different operating systems (OS) including Linux, Mac OS, Windows, vxWorks, and others. The RDB can also be based on different implementations (ORACLE, MS SQL, etc.). Expanding on a traditional approach, each combination of OS/RDB would require a special RDB driver for each OS. These multiple combinations create configuration control, maintenance, and portability issues.

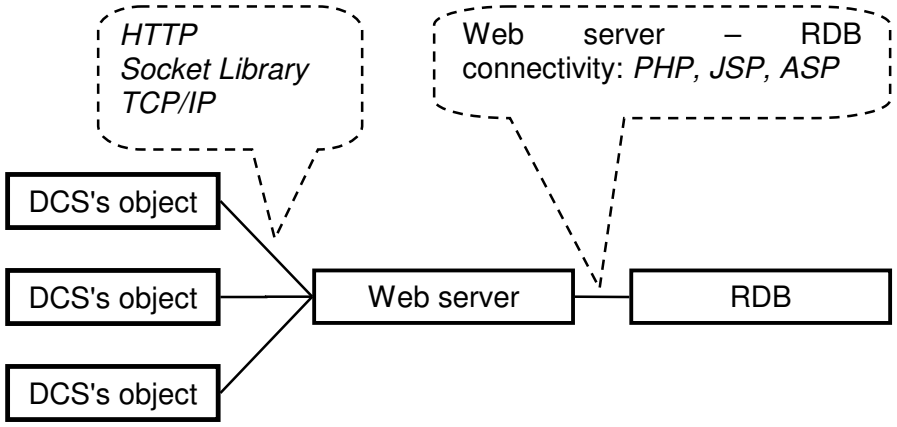


Figure 2 Solution structure

A simpler solution (Figure 2) adds a web server between the DSC's object and the RDB. The web server can be any suitable OS (Linux, MAC OS or Windows). With this solution there are two interfaces. The first is the DCS's object to the web server interface. It is based on TCP/IP protocol. The TCP/IP protocol is supported by the socket library on any OS. Using the HTTP protocol on top of Socket library expands the interface and allows the connection to the RDB. The second interface is from the web server to the RDB. This interface is a standard solution. The bundle of web server and database server is widely used in different web applications such as online catalogues, electronic commerce, etc. There are many implementations (including open source) of this interface. The most standard are Apache web server + PHP module talking to MySQL database. Java-oriented developers can use Apache Tomcast web server with JDBC drivers for any Java enabled database. The Windows people can rely on Microsoft .NET technology. In all cases, specific implementation is developer choice and doesn't affect the DCS side, so migration between different databases and/or web servers is fairly easy.

3 SNS IMPLEMENTATION

SNS has deployed approximately 500 IOCs within its control system. And although many of these IOCs are similar in the hardware configuration, each of them has a unique purpose. An IOC's uniqueness is established based on its software and specifically on certain files that are read during the boot process. Because the SNS control system is based on the Experimental Physics and Industrial Control System (EPICS), we use the EPICS standard configuration files (st.cmd, *.substitutions, *.db, etc). These files are used on our traditional IOC implementations, and they impart the IOC purpose. We also have deployed IOCs that use LabView to create the EPICS configuration files. These LabView-based IOCs use a single text file to establish uniqueness. All of the files discussed here can be referred to as configuration files.

To test the web server-to-RDB solution, it was decided to use it as a means to retrieve RDB data, specifically the configuration files for EPICS IOCs. As said above, during the boot process, the IOC uses the configuration files to form the IOC's identity. These files are usually stored on the IOC's hard disk. It was decided to develop a protocol to support file operations with ORACLE RDB, Web Server, and IOCs. The RDB stores these required files. Now the initial process during boot is the communication with the web server and consequently the RDB causing the transfer of required files to the IOC's hard disk. After the transfer, the IOC continues its usual boot process. Files can be manipulated at the engineer's convenience and then implemented with an IOC reboot.

4 RDB TABLE IMPLEMENTATION

The SNS RDB is based on ORACLE and is extensive in its schema design. To increase our flexibility with the configuration files, quite a few tables were added to the schema. These new tables give us version control and rollback functionality, but the main table used with this solution has columns that capture the IOC name, file name, file path, and file contents. This table is updated via stored procedures and triggers within the database. This allows the data that is contained within a configuration file to be controlled and easily manipulated but gives the web server-to-RDB protocol a single place to retrieve an IOC's files.

4.1 Software

The IOCs are modified to use a client library named HttpClientLibrary.c, which is based on an EPICS socket library (from EPICS Com library). The library should work on any EPICS-supported OS and has been tested with Windows and vxWorks.

Implementation of the IOC to use the web server protocol on the Windows platform was done

with the use of software already used at SNS. Originally, a C library (HttpClient.dll, EPICS com.dll) was used. This worked well in our test environment but was dropped because the implementation of the library was difficult. The windows-based IOCs vary in both version of LabView and EPICS. Maintenance of the library was complicated, and this was compounded with different version combinations. An independent LabView virtual instrument (VI) was written as a result. This VI, HttpClient.vi, was written using the pure LabView internal socket library. This one VI can be modified for any version of LabView and then called as part of the boot process of any LabView-based IOC.

A nice extra with the LabView-based IOCs is that they can be triggered via an EPICS process variable (PV) to download a new configuration file without rebooting. The engineer that is modifying a configuration file uses the tool that interfaces to the database to also modify a PV. This PV is monitored by the IOC, and when the IOC sees the change, it implements the HttpClient.vi to get the new configuration file and imports its new configuration on the fly.

To enable this functionality in vxWorks, the library is loaded by inserting the following lines after "< cd Commands" in the st.cmd file.

```
hostAdd( "RDBServer", "172.31.75.144")
hostShow()
iFGetFilesByHTTP("RDBServer:8080")
```

4.2 Process Definition

To download the IOC required configuration files, a simple progression is followed. In short, the IOC initiates a request to the web server. The web server determines the number of files required for the specific IOC, and then through a second request downloads the files from the RDB. The IOC then receives the files from the web server.

We use a simple, pure java web server implementation. Although any commercial web server could be used, we found that having a small java server gives us some nice tools to troubleshoot any problems. This web server is capable of processing html, graphics, and text files. While processing these files, the server reads the content of the file and substitutes variable names with parameters provided in the URL string and runs the result string as a database query. The web server returns the query execution result in a plain text format.

Initializing the process, an IOC requests the web server to "*GET /FileList.sql"*. The IOC's request tells the web server to create a call to the RDB for a summary of file information specific to the IOC. The web server request is a simple SQL statement or something like "select config_file_id, config_file_name, config_file_location from ioc_file_table where IOC_network_name = '$(IOC_NAME)"'. In this query, we have one parameter $(IOC_NAME) that is substituted by the client hostname or as a URL string parameter. The returned list of files contains lines (one line per file) separated by <CR><LF> (this is critical for LabView). The IOC then requests the files using the file ID, one by one following the previously returned list. The file request query is "select contents from ioc_file_table where config_file_id = $(file_id)".

As one can see, the IOC deals with two types of requests and has no idea that the results are coming from an RDB. It is easy to modify the web server to serve the files from any other sources without any change at IOC level.

Testing of the web server and RDB can easily be done. Using an internet client, like Internet Explorer, different URLs can be entered to query the RDB. From the browser on an IOC, the URL "http://ics-srv-test1.sns.ornl.gov:8080/FileList.sql" can be used to retrieve the list of files available in the database. The URL "http://ics-srv-test1.sns.ornl.gov:8080/GetFile.sql?FILE_ID= 295" is an example of a request for the contents of the file with the ID of 295. If an IOC doesn't have an internet client, any other computer with an internet client can be used to test. Using an URL with a suffix, like "?REMOTE_SHORTNAME=pc62273", the database is queried for data related to the network name. Add in the suffix, in this case "pc62273".

4 CONCLUSIONS AND FUTURE PLANS

The protocol that allows an IOC to access our RDB via a web server has given us a powerful tool and the ability to take advantage of its functionality in many ways. We have taken advantage of a central storage area for our configuration files, and it has more than just saved us time. We now have the ability to use standard RDB tools to manipulate the data contained within the files. We have also expanded the structure of the RDB to track data related to usage of the files. The use of this protocol also lends itself to uses beyond configuration files. The SNS database houses data related to all of SNS and any computer can use these libraries to retrieve relevant information.

Control System Using FL-net for Communication between Different PLC

Akihiro Osanai
Kyoto University, Kyoto 606-8501, Japan

Abstract

The control system using PLC and LabVIEW on PC is constructed for an FFAG accelerator complex in KURRI (Kyoto University Research Reactor Institute). Considering to build another control system into a current system in future, we contrived the system having two-type PLC (FA-M3 and MELSEC) connected each other through FL-net, which is an open network supporting cyclic data transfer. We introduce a successful example in the case of applying the system to the ion source of the FFAG accelerator.

INTRODUCTION

The FFAG accelerator complex in KURRI has a control and interlock system constructed by PLC (Programmable Logic Controller) and LabVIEW. PLC and LabVIEW are used as low and high level sequence controller respectively. For convinience of making sequencing programs and maintenance, the system was unified by using same vendor PLC, FA-M3 manufactured by Yokogawa Electric Corporation [1]. As planned extension of transport beam line, we might need to install devices controlled by the ohter vendor PLC in future. Figure 1 shows the schematic diagram of the control system consisting of multi-vendor PLC. In that case, we will face problems of response to the different interfaces at high level control. To avoid that, one solution is replacement of PLC with our familiar FA-M3. We had an idea making different vendor PLC connected and share their data each other through FL-net, Installing mediating PLC, interface conversion is implemented at high level control without any changes at low level sequence control.

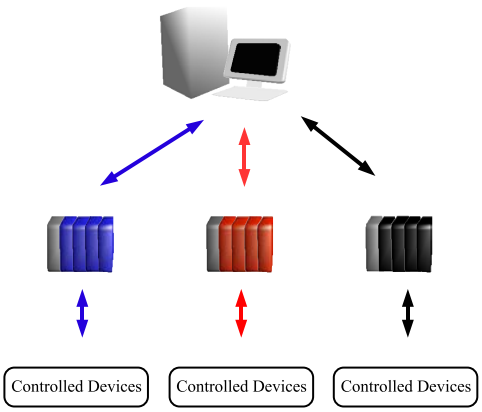


Figure 1: Schematic diagram of the control system constructed by multi-vendor PLC. This system is required adaptations to multiple interfaces of PLC at high level sequence control.

The FL-net, which was initially developed by MSTC (Manufacturing Science and Technology Center), was transferred to JEMA (Japan Electrical Manufacturers's Association [2]) and standardized as the controller level network. Following is main features of the FL-net.

- Open network, which realizes communication among control devices
- Physical layer is Ethernet.
- Master-less and token bus method is adopted.
- Cyclic transmission is available.
- Common memory is constructed on the virtual memory space.

These features helped us adopt the FL-net. In the nature of the control system, high speed and large quantity data transmission, reliability of the system and high cost performance were required. Ethernet cables such as 10/100BASE-T/TX are available as transmission media. Master-less method prevents failure of the specific node from causing system down. Communication performance is guaranteed by token bus method and cyclic transmission. Data is refreshed within 50ms among 32nodes. Figure 2 shows a schematic diagram of the control system using FL-net. Sharing data on the FL-net, the interface between PC and PLC is unified.

To confirm this system working out well, we applied it to the ion source of the FFAG accelerator. In this paper, we describe the details and successful result.

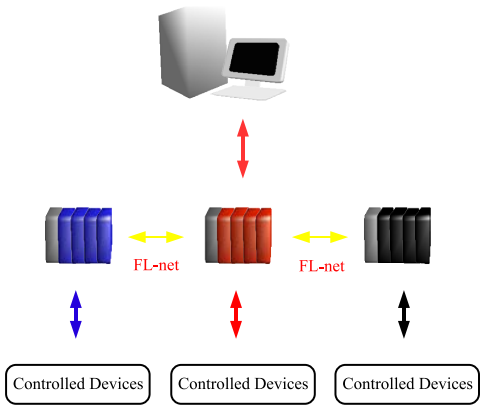


Figure 2: Schematic diagram of the control system using FL-net. This system keeps unification of the interface of PLC at high level sequrence control.

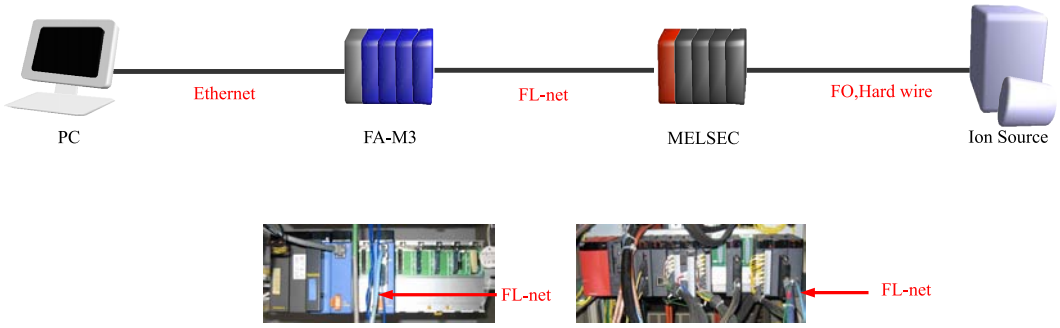


Figure 3: Framework of the ion source control system using FL-net.

PREPARATION

Hardware Connection

The framework of the ion source control system is shown in Figure 3. MELSEC manufactured by Mitsubishi Electric Corporation [3] was used for the ion source as PLC. To prevent from the electrical noise and radiation damages, main PLC including CPU and FL-net module was connected through fiber optics cable to slave module blocks controlling power supplies on high voltage. The FL-net module of MELSEC communicated with that of FA-M3 through exclusive Ethernet cable for FL-net. FL-net can not be mixed with conventional Ethernet. Mixture of the network would cause serious communication failure. Operators could control the ion source using control software on PC through FA-M3 on Ethernet.

FL-net Setting

The FL-net setting on PLC is very easy. Local IP address on the FL-net and common memory should be set. The last 8bit of the IP address corresponds to number of the node. We allotted 192.168.0.1 and 192.168.0.2 for FA-M3 and MELSEC respectively. Common memory space is virtual memory space that can be accessed by all connected nodes. Allocated data area of each PLC as common memory space, it looks like a single PLC viewing from high level sequence. The concept of the common memory is shown in Figure 4, and an example of register allocation is shown in Table 1. Buffer memory (MELSEC) and link relay (FA-M3) were used as common memories.

Software

LabVIEW was used as a software for communication with PLC and high level sequence control. The software has powerful graphical development environment for measurement and control program. We can use it without knowledge on programming and special training.

PERFORMANCE

The control system was performed linking PC by both connections, wired and wireless. Cyclic period for reading data was set 200ms on LabVIEW. The control monitor on PC is shown in Figure 5. Setting parameters are in white boxes and the responses are in yellow. The graph on the picture is real-time chart of acquired data. In continuous operations, system worked out well under this circumstance. Futhermore using the software on the portable PC, through wireless LAN, we could control devices some time in the control room and another time in the accelerator's room for test operation.

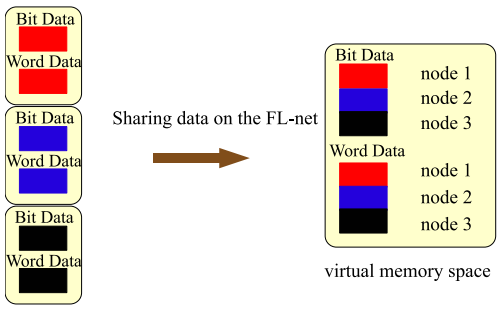


Figure 4: Concept of common memory. A single PLC is constructed on the virtual memory space.

Table 1: Example of Register Allocation

node1(Bit) FA-M3 → MELSEC	Buffer Memory (MELSEC)	Link Relay (FA-M3)
H2 valve open	0x1C00(01)	L0001
mass flow remote	0x1C00(11)	L0011
50kV output on	0x1C01(01)	L0017
Arc output on	0x1C01(11)	L0027
Arc mode auto	0x1C01(13)	L0029
master start	0x1C02(01)	L0033
master stop	0x1C02(02)	L0034
master reset	0x1C02(03)	L0035
100kV output on	0x1C02(11)	L0043
bias remote	0x1C03(01)	L0049
bias output on	0x1C03(03)	L0051

Figure 5: Capture of control monitor on the PC

SUMMARY

The control system using FL-net has been easily installed (and it is still applied to operation of the ion source). This will help unification of interfaces at high level sequence without any changes at low level sequence.This achievement could allow flexibly to build system that is independant of PLC vendor.

REFERENCES

[1] http://www.yokogawa.com/

[2] http://www.jema-net.or.jp/English/

[3] http://global.mitsubishielectric.com/

COMPACT MONITORING AND CONTROL SYSTEM WITH EVENT SIMULATING

Vinogradov V.I.

INR RAS, Moscow, Russia.

.

Abstract

Problems of event recognition, registration and analysis in real time with embedded modeling and simulating event signals are general for many applications. Proposed effective compact system architecture with embedded SBC and DSP-based measurement and control modules can be used effectively as autonomous station or terminal node in distributed network for monitoring, registration and control system.

1. Introduction to Event Registration

Event recognition and analysis systems are connected with general problems of monitoring and registration signals for data analysis and control in real time. Any events are abnormal signals recognized by system on special algorithms in real time as critical situations with monitored objects. Special front-end electronics are required for any applications. The nature of any objects (engineering or biomedical) is no means, becouse all situations are reflected in specified signal processing algorithms. It can be an object, which critical status reflected by some number of abnormal object signals. Recognition of events on many periodical signals can be done by minimum or maximum limits of each normal signal or by special addition signal forms analysis of abnormal signals. At the same time system should select signals from any noise. As regards to engineering objects events should be recognized in very short time limits and registration should be done during time interval before and after the events occur. Many signal recognition and registration algorithms can be described by some number of control parameters in software.

Event registration with modeling and simulation should be executed in compact modular system, working in real time. An object critical situation can be described as the event by a number of abnormal signals.

2. Compact modular RT-system Architecture

Recognition Event parameters provide discovering of the event signals. All of these signals can be registered from real object in real signal data files and collect in file server for analysis. At the same time all this events can be modeling and simulating in the same instrumentation system to study and predict any critical state of engineering objects. Collection of real and modeled events in server can be used as knowledge base for future prediction critical state and for optimal control of the engineering objects. In this case simulated events should be registering in the same instrumentation systems as monitoring events.

Each node should content SBC and DSP-based subsystem modules for measurement and control in real time. All interconnected compact terminal nodes should work in Distributed Network according to real time requirements. Required numbers of compact nodes can be used for monitoring, registration and analysis more complex distributed engineering event signals. Supervisor station as the central node in Distribution system can be used for modeling and simulating event signals in autonomous mode or for distribute simulated parts on selected terminal nodes fro simulating complex event and registered them in he same environment.

Embedded SBC modules for open compact system architecture can be based on different bus architectures, including VME/VXI and cPCI/PXI, which formats offers some advantages. Compared to VME (3U), cPCI (3U) are more cost effective systems. The backplane approach makes maintenance and upgrading of 3U cPCI modules simpler. The cPCI/PXI bus supports 32-bit or 64-bit data transfers in both single- and double-wide boards. The cPCI's (3U) bus performance is superior to 3U VME. The cPCI/PXI enables system flexibility extending the PCI slot limit from 4 to 8 cards. The cPCI for industrial fields (like VME) and PXI – for modular Instrumentation Systems (like VXI) are based on passive backplane. The cPCI/PXI boards support I/O for Industry automation, which requires distributed I/O, and the Field Buses for DAQ and Control, monitor, and report on processes, but have high price.

Proposed Modular strategy concept in system design is based on compact RT-system core with effective modular structure and minimal I/O channels for registration of signal waveforms in real time, discovering some events as abnormal status of object and registered data before and after the events occurred according to special algorithms.

Each compact RT-system node for monitoring and control can be effective (performance/ price) for using with up to 8-16 input signal channels and two analog output channel, up to 32-64 digital signal I/O and should be based on typical SBC and DSP-based measurement and control modules. The modular approach is based on embedded passive 3-4 PCI slots bus for low-power SBC and 1-2 DSP-based modules, embedded in the box. Collected real waveform signals as the events in files for analysis can help to discover any risk situation in complex environment and use optimal control algorithms. The RT-system as core network node includes embedded passive 3-4 PCI slots passive bus with SBC and DSP-based typical instrumentation modules. Control functions include switching off some part o object (equipment) at the moment of event occur.

Effective (performance/cost) very compact modular system today can be constructed on passive PCI bus with only 3 PCI slots for SBC and DSP-based measurement and control nodules. Such system can be embedded in special box with signal conditioning and used as autonomous or as terminal system (node) in distributed network architecture. Multi channel analog I/O can be used for simulating of the signals modeling by software and for registration on all input channels as real event signals for analysis. Dimensions of SBC – 185*122 mm, power supply - 5V (1.8A). Supervisor node has full functions as Terminal node in autonomous mode or can be used as central supervisor node with central signal generator functions. Autonomous system includes event signal registration, control, simulation and visualization.

Modeling and simulating tasks should work in the same compact modular system in real time. Such system can be used as autonomous monitoring and control station with embedded modeling and simulating event signals. Complex events can be connected with many distributed objects. For these problems many compact computer-based nodes should be integrated in single Distributed system on the base of switched network architecture.

3. Embedded Modeling and Simulating

A lot of basic RT-system works as Control terminal station (TS network node) can be used in the Distributed scalable system for monitoring and control of complex objects. The instrumentations can be used also for signal events modeling and simulating. Engineering event signals are described on examples of Industrial Engineering object signals. There are different monitoring and control devices in energetic, including signal and event registration, relay protection, signal modeling and simulating. They shipped usually as independently black boxes for professional users, which required open systems to analyze real situation with object status to construct special optimal algorithms for event discovery and control. Now it is possible to construct the system jointing registration and control functions in a single RT-system, monitoring and registering abnormal signal forms before and after the event moment for analysis and control. Algorithms to discover the events can be constructed in the same instrumentation system and tested with special object environments. This compact modular approach can be used also for any distributed modular systems with multimedia real-time requirements (audio, video) transmissions and control on the base of network switched architecture for real time applications.

Embedded modeling and simulating event signals can help in construction and testing special new algorithm for optimal control in the same instrumentation system. According to constructed model simulating by internal generator the node can register periodical signals with event modeling and display them on the monitor for analysis the same way as real signals and events registering. Special testing of all analog and digital channels can be done by switching input, testing and simulating signals in special front-end signal condition device. Embedded Modeling and Simulating in Distributed system can be based on parallel using of selected number of TS for distributed complex event. Each TS has embedded DSP-based module with DAC for simulating event signals. For high-precision simulating should be used special independent

3. Embedded Modeling and Simulating

A lot of basic RT-system works as Control terminal station (TS network node) can be used in the Distributed scalable system for monitoring and control of complex objects. The instrumentations can be used also for signal events modeling and simulating. Engineering event signals are described on examples of Industrial Engineering object signals. There are different monitoring and control devices in energetic, including signal and event registration, relay protection, signal modeling and simulating. They shipped usually as independently black boxes for professional users, which required open systems to analyze real situation with object status to construct special optimal algorithms for event discovery and control. Now it is possible to construct the system jointing registration and control functions in a single RT-system, monitoring and registering abnormal signal forms before and after the event moment for analysis and control. Algorithms to discover the events can be constructed in the same instrumentation system and tested with special object environments. This compact modular approach can be used also for any distributed modular systems with multimedia real-time requirements (audio, video) transmissions and control on the base of network switched architecture for real time applications.

Embedded modeling and simulating event signals can help in construction and testing special new algorithm for optimal control in the same instrumentation system. According to constructed model simulating by internal generator the node can register periodical signals with event modeling and display them on the monitor for analysis the same way as real signals and events

registering. Special testing of all analog and digital channels can be done by switching input, testing and simulating signals in special front-end signal condition device. Embedded Modeling and Simulating in Distributed system can be based on parallel using of selected number of TS for distributed complex event. Each TS has embedded DSP-based module with DAC for simulating event signals. For high-precision simulating should be used special independent generator nodes or module at supervisor station (SS node). This event signal with or without noise can be summarized with a periodical normal signal in each channel to get any form of simulated events.

Proposed modeling event signal approach is based on general trapezoidal event signal construction, which can be transformed in required signal forms. Each part of the basic signal form can be added by asymptotic signal form with the same time parameter to get any real signal forms. The constructed event signal can be added to normal sinusoidal signal with some additional noise to get required event model.
Signal on each channel is simulated independently as real object event. There are front-end object oriented signal condition electronics to switch all input and output channels for measurement, test and simulating modes. All simulated event signals and typical real events for the some complex objects can be compared, analyzed and used for predicting and control at the moment of critical situation in environment is occur.
Distributed modeling and simulating of object signals can be done in each node parallel as in autonomous mode with monitoring by SS-node, using the same input channels. Modeling and analysis event signals can be collected in central node (file server or DB) with real signals for analysis. Additional simulating functions of TS and SS-nodes are system testing with real objects signals produced by internal embedded signal generator or by external Node (SS with precision DAC). Future system development can be based on serial interconnect for high performance modular systems, integrating on the base of new technologies (WLAN, PLC).

Proposed embedded modeling and simulating event signals in the same instrumentation system provides construction of new algorithms for recognition and predict some critical situation in open system architecture. Autonomous and switched based system architectures for event registration and analysis with embedded modeling and simulating are proposed and discussed.

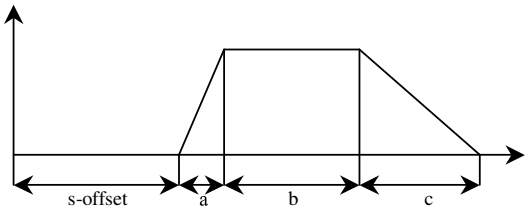
SUMMARY

1. Compact Modular systems on the base of 3-4 PCI slots passive buss can be effective platform for Monitoring, Registration and Control Systems and can includes USB interface for effective serial connections to measurement and control modules.
2. Autonomous system consists of SBC and DSP-based measurement and control modules and can be used as the terminal nodes in network based architecture with Supervisor Control station.
3. Interconnections to supervisor node and server is based on network switch.

REFERENCES

[1] V.I.Vinogradov. Event simulating in object analysis system. EUROMEDIM2006. CERN Conference. Marseille, France.
[2] V.I. Vinogradov. Modular RT-systems and networks for Monitoring and Control with event modeling and simulating. Preprint INR RAS -1163/2006/.

1. Compact Modular systems on the base of 3-4 PCI slots passive buss can be effective platform for Monitoring, Registration and Control Systems and can includes USB interface for effective serial connections to measurement and control modules.
2. Autonomous system consists of SBC and DSP-based measurement and control modules and can be used as the terminal nodes in network based architecture with Supervisor Control station.
3. Interconnections to supervisor node and server is based on network switch.



Event Signal is constructed from basic trapezoidal form with a, b, c, d – parameters:
1) a = b = c = 0 – no signals;
2) b = c = 0 - linear front saw signal;
3) a = b = 0 - reversed front saw signal;
4) b=0 - triangle signal with c, b;
5) a = c = 0 - rectangle signal; with any b;
Where s – signal time shift (offset),
T = a + b + c - is duration of basic event,

Figure 1. Basic model of Event signal for modeling and simulating.

NEW HIGH-PERFORMANCE MODULAR COMPUTER SYSTEM ARCHITECTURES FOR CONTROL NETWORK APPLICATIONS

Vinogradov V.I.*

INR RAS, Moscow, Russia

Corresponding author. Tel.: 7-4967514631 mail: vin@inr.ru

Abstract

Convergence of computer systems and communication technologies are moving to switched high-performance modular system architectures on the base of high-speed switched interconnections. Multi-core processors become more perspective way to high-performance system, and tradition parallel bus system architectures (VME/VXI, cPCI/PXI) are moving to new higher speed serial switched interconnections. Fundamentals in system architecture development are compact modular component strategy, low power processor, new serial high-speed interface chips on the board, and high-speed switched fabric for SAN architectures. Overwiev of Advanced modular concepts and new international standards for development high-performance embedded and compact modular systems for real time applications is described

1. Introduction to bandwidth problems.

The convergence of computer and communication to distributed modular systems integrating different types of information like *data, audio, video* meets real time requirements as RT-systems for DAQ and Control applications. Next generation 32/64-bit processors coming to frequency limit, and advanced multi core processor architecture becoming main way to get more higher performance. Interconnect I/O peripheral subsystems to processors are coming to his limit also, and new problems for high-speed interconnect appeared. Ethernet, USB, and PCI buses use shared parallel bus architectures no longer have adequate bandwidth. **A new serial interconnects** revolutionize next generation high-performance computer systems, including migration from PCI to PCI Express and migration from ATA to Serial ATA in telecom and datacom systems.

2. Requirements to new computer systems

High performance computing (**HPC**) encompasses advanced computation over parallel processing, enabling faster execution of highly compute intensive tasks such as climate research, molecular modeling, physical simulations, engineering modeling and simulating. cryptanalysis, geophysical modeling, automotive and aerospace design, financial modeling and data mining. The tradition bus architectures proved to be too slow for fast graphics cards and an additional bus, the accelerated graphical port (AGP) was introduced. PMC mezzanine sub-modules for PCI modules as additional function extensions inside some modules were developed as industrial standard.

Real-time requirements are applied not only to DAQ and Control, but also to new Distributed data transfer, telecom and datacom systems including picture processing, audio and video. Some applications are significant consumers of high-speed peripherals with high-speed interconnects such as DVI, SATA and 10Gbit Ethernet; unmanned vehicles place significant size, weight and power constraints on their electronic subsystems; and 2D/3D image processing creates huge demands on systems for capturing, collect and interpreting information. Radar signal processing has required massive bandwidth more than offered by bus-based systems; software. A revolutionized approach is required to decide all today's problems for high-performance system. Parallel data processing, modeling, simulation, and image processing in real time are compute and communication intensive. Compact computer systems and distributed switched based network architectures will be required as next generation systems in different applications.

System Area Network (SAN) architectures were developed on the base of Serial Interconnect for high-performance data processing and mass storage access.
One of the first point-to-point Interconnect for multiprocessor was Scalable Coherent Interconnect (**SCI**) like Fiber Channel for storage A number of computer industry leaders decided to design the future I/O subsystem, and advanced system architectures were developed for computer industry, telecom and datacom applications. Among modular computer systems PCI and VME bus architectures are most popular in industry today. The requirements led the industry to standardize creation the *new serial interconnects* in open system as well as to extend the useful life of tradition PCI and VME-based systems. New standards are primarily the concern of two industrial specifications PICMG and VITA.

3. Serial cPCI Express interface

Two competing initiatives started, one - the Next-Generation I/O system (Dell, Hitachi, Intel, NEC, Siemens and Sun Microsystems), other - the Future I/O system (Compaq, IBM, and Hewlett Packard). Both groups decided to unify their efforts by bringing together the best ideas of each of the two separate initiatives. The result was new serial PCI interface. Many applications *like image processing, biological, medical technologies and robotics* depend on rapid data transfer between 32–bit CPU and various peripheral devices. Such connections are still realized mostly via a PCI bus. Next **PCI-X** specification allows for a 64-bit version of the bus operating at the clock rate of 133 MHz, but this is achieved by easing some of the timing constraints: system can have only 1 slot on the bus, 2 PCI-X slots allow a maximum clock rate of 100 MHz, and 4 slots would drop down to 6 MHz. **PCI** architecture was adopted for telecom industry as compact PCI **(cPCI)** with maximum data throughput from 133 MB/s then moving to 533 MB/s
Serial PCI Express (PCIe) is transparent on *physical layer* to application software, and programs written for traditional PCI devices can run on PCIe devices. Data is transferred via high-speed, point-to-point serial links know as lanes. PCI and PCIe can be used together in the Hybrid system. Each lane comprises a pair of differential conductors with 250 MB/s pro direction: one pair is used for data transmission, other - for receiving. These lanes can be bundled to a maximum of 32 lanes per channel (up to 16 GB/s in both directions). PCIe bandwidth is scalable. The common lane configurations are x1 (“by 1”), x4, x8 and x16. The bandwidth available is proportional to the number of lanes. Typical 64-bit PCI-X bus requires 127 signal pins on multiple board layers versus x4 PCI express slot that provides twice as much bandwidth and only requires 15 signal pins. PCIe replaces shared bus with a shared switch.

Compact PCI Express (cPCIe) supports bus structures with 4 to 16 lanes per channel enabling data transfers up to 4 GB/s, which bi-directional, doubling the 250 MB/s data throughput to 500 MB/s. If cPCIe board cannot administer all available lanes of the board, then the unused lanes will be automatically deactivated during initialization. Remaining from the current cPCI specification is the power connector for the PSU and the slot for parallel-bussed peripheral modules (PICMG 2.0).

4. Compatible VME bus bandwidth extension

VME bus systems have been used in all segments of science), industrial and military applications. There are a few reasons for parallel compatible VME bus extension: 1) Interrupts over SI is not best decision for RT-systems. 2) Arbitration between peer-to-peer or priority sequenced tightly coupled processors is similar as interrupt processing. **VITA** is aimed to increase VME bus performance, while maintaining backword capability. The key efforts are: faster parallel bus (VITA 1.5); multi-Gigabit switched serial interconnects (VITA 41 and 46) and new mezzanine cards (VITA 42). The first VME bus renaissance was being the 2eSST protocol, which implemented with chips from Tundra and Thales, enables bus to run at **320 MB/s (**an 8x bus performance of VME64’s).
The VITA 41.x family of specifications is VME bus extensions for serial switched technology**. VXS (VITA 41.0)** defines physical features that enable high-speed serial links in a VME bus –compatible system with the addition of high-speed connector to the VME64x board in the P0/J0 position. The VXS backplane currently has Infiniband (VITA 41.1), RapidI/O (VITA 41.2), Gigabit Ethernet (VITA 41.3) and PCI Express (VITA 41.4) protocol layer**. Serial VITA 46** replaces all the DIN connectors on a VME board with high-speed connectors supporting signaling rates up to **6.25 Gbit/s.** There are a series of standards, including VITA 46 **(VPX)** and VITA 48 **(VPX RDI).** VITA 46 standard provides **4** switch fabric ports of **10 Gbit/s each** in the initial configuration, and the capacity for more than 20 ports in fabric-only configurations. The specification expands users I/O capability and provides a broader ability to map high-speed user I/O or fabric connections from the emerging XMC (VITA 42) mezzanine to the backplane. At the heart of VPX is a high-speed backplane 7-row MultiGig RT2 connector (signal rates up to 6.25 Gbit/s), developed by Tyco.. It provides a effective way of allowing VME bus users to leverage the performance of the high-speed switched fabrics such as **StarFabric, RapidIO and PCI Express.** VSO was developed specification for enhanced design (VITA 48.x), electronic cooling (VITA 50), reliability predictions (VITA 51), lead-free practices (VITA52), technical management (VITA 53) and other requirements. Compatible VME bus bandwidth extension
Provides high-performance increased in many times compared to ordinary VME bus systems.

5. SCI Interconnect and switched system.

Scalable Coherent Interconnection **SCI** was one of the first serial interface developed by researcher (Stanford University) and using from 2002 with PCI and cPCI modules and with high-speed switches..
New version of SCI-PCIe interface modules and switch are modern Interconnect for High-performance SAN with Distributed memory architecture. SCI was developed as standard for high-performance multiprocessor with gigabit parallel-pipeline data transfers. The first commercial supercomputer was developed by Sequence with quadrant nodes interconnected by SCI. Interconnect modules and switches can be used in a variety of topologies including rings or switching 2D torus. The PCIe based SCI Adapter Modules - D35x family are modern high-performance solution for server systems, HPC clustering and embedded applications. They are components for building a high performance multiprocessor, Workstation and Server cluster configurations for many applications. The D351 offers 10 Gbits/s link speed (bi-directional) that makes it an ideal platform for moving large volumes of data from system to system. For higher bandwidth the D350 and D352 use two bi-directional links effectively doubling the throughput to an amazing 20 Gbits/s. The ultra-low 1.4 microseconds application-to-application latency reduces overhead of inter-node control messages, makes D35x modules a good choice for RT-systems. Two other versions of the **cPCI-SCI** modules are available also to support one or two SCI rings, which have distributed switching capability offering the ability to design large high performance clusters. The cPCI-SCI Adapter Card (module) is compatible with the standard PCI-SCI and PMC-SCI Adapter Cards and the 8-port SCI switch. SCI distributed memory architecture enables applications to take full advantage of the new 64-bit processors (64-bit AMD Opteron), which has been developed around Dolphin's Software Infrastructure for SCI (SISCI) - an API library, that enables applications to use clusters and reflective memory. Typical latencies for PCI bus architectures are 1.4 microseconds for an 8-byte buffer store and 3 microseconds for a 512-byte store. SCI has excellent bandwidth capabilities in direct memory access (DMA) and remote memory access (RMA) mode.

6. Infiniband switched system architecture

The computer industry leaders (Compaq, Dell, Hewlett-Packard, IBM, Intel, Microsoft, and Sun Microsystems) formed the InfiniBand Trade Association (ITA), which formed IT Program (2001) to provide the InfiniBand architecture specification.
The InfiniBand specification (2001) defines the interconnect architecture that will pull together the I/O subsystems of the next generation servers. It is industry standard technology that advances I/O connectivity for high performance computing clusters, breaking through the bandwidth and limitations of the PCI bus by migrating from the traditional shared bus architecture into *switched fabric architecture*, where two or more nodes are connected to one another through the fabric. The architecture is based on a serial switched fabric defining link bandwidths 2.5 - 30 Gbits/sec, resolves the scalability, expandability, and fault tolerance limitations of the shared bus through using switches and routers in the switch fabric.
InfiniBand switch fabric, consisting of a single switch or a collection of switches and routers, was developed as a System Interconnect platform for modern servers, Data Centers, SAN, HPC, storage subsystems and virtualization**. Server clusters** and grids, linked with high-speed interconnect, creates intensive compute power solutions. With it's scalability and efficiency small and large clusters scale up to thousands of nodes. With 20 Gb/s node-to-node and 60Gb/s switch-to-switch solutions available, and a roadmap to 120 Gb/s, Infiniband Adapter 10Gb/s matches Gigabit Ethernet pricing. High speed Channel Adapters (**HCA**) can support multiple end-points that can provide dedicated granular QoS and security services to virtual servers, storage, HPC and management applications.

7. Advanced TCA and MCA modular systems

Advanced Telecom Computing Architecture **(ATCA)** base specification defines the form factors, core backplane fabric connectivity, power, cooling, management interfaces, and the electromechanical specification of the carrier board (the existing IEC 60297 Eurocard). ATCA compared with existing TCA backbone systems, is faster, fault-tolerant and easier to service with switch fabric, system management, hot swap and modular format. The carrier can be a simple passive board or SBC.
The backplane has become the core of interconnection between each of the modules.
Data have to be switched and transferred at multi-gigabit speed and designer must consider connectors as part of the signal transmission line and take care of impedance, delay, skew, and crosstalk Cable interconnects to the backplane and Mezzanine Card connectors become the system infrastructure. Differential signaling requires 2 separate lines for each signal with lower voltage, but it offers greater isolation from noise. Differential pair signals must arrive at their destination at the same time. Since characteristic impedance is a function of geometry and materials, each of these variations can alter the impedance and generate reflections.

PICMG 3.x specification series define how to map a specific switching interconnect technology onto the physical framework. PICMG 3.1 defines the mapping of Ethernet and Fibre Channel (FC), PICMG 3.2 defines Infiniband, PICMG 3.3 - StarFabric, PICMG 3.4 - PCI Express, and PICMG 3.5 - RapidI/O.

PMC modules for cPCI permit application customization but do not meet some of the telecom requirements (hot swap). The evolution of PMC, **XMC**, addresses the connector signaling issue, but it lack the required I/O pin count. Advances Mezzanine Card, **AMC** settled on a carrier boards into ATCA systems has lower pin-count connector geared towards switch fabric and provides increases scalability and flexibility in system development. An AMC design key is hot swap. AMC is deigned to meet the carrier grade needs of reliability, availability, and service ability (RAS).

Target interfaces are High-speed mezzanine AMC, which optimized for, but not limited to, ATCA carriers by PICMG for targeted interfaces like PCIe, Advanced Switching and Gigabit Ethernet.

AMC.0 – is common specification (mechanics, management, power, thermal, and interconnect).

AMS.1 specification defines the implementation of PCIe and Advanced Switching,

AMC.2 adds Ethernet interfaces and

AMC.3 adds specific storage interfaces such as FC. Depending on the number of the contacts one differentiates between AMC module half height (HH) and full height (FH) as well as single width (SW) and full width (FW). The AMC architecture is flexible and supports a number of transfer protocols with different bandwidths.

They include 10G Ethernet (IEEE 802.3ak – CX4), 10G Fiber Channel, serial attached SCSI (SAS), Serial ATA2 (SATA-2) and InfiniBand

Advanced Micro TCA (2005, 2006) is developed now as new advanced compact modular system standard. It based on the mezzanine format AMC, which could be used in advanced low-cost compact modular systems with own backplane. Advanced MicroTCA will be successful in many types of application - both in telecom and in computer systems and in such fields, as the biology, physics, medicine, engineering, robotics and military. The overall concept allows AMC modules to be plugged directly into a system backplane. If the processor, north-bridge, south-bridge, memory and flash of a PC are modularized in a compact, highly integrated fashion, it is Computer-On-Module **(COM).** By segmenting the processor complex onto a COM, designers can focus on to carrier board, which can be paired with COM modules to achieve required performance and move to the next generation system.

MicroTCA is a draft specification that gained status as a subcommittee subject under the PICMG (2004), which release COM.0 (COM Express) to use PCIe in COM.

ETX and ESB standards were also suggested in telecom, each had meeting the requirements for a next generation COM systems. COM Express builds upon the ETX specification (Kontron). Digital Signal speed in PCB is moving up to 5 Gbit/s.

COM XTX standard replaced ISA bus with PCIe technology, SATA and additional USB 2.0. XTX enhances PCI by 4 PCI express lanes, which offer data throughput 10 times of single 32 bit PCI bus.

Embedded Computing companies see expanding opportunities in defense and commercial-off-the-shelf (COTS) focused on industrial automation, healthcare equipment, communications, automotive electronics and other fields.

Advanced Micro TCA is most compact modular system architecture with AMC modules installed directly in the crate. These high-performance systems can be used effectively in many applications, including monitoring and control terminal systems, Data Acquisition and Processing, network and telecommunications, medicine and biology, multimedia (audio and video) applications and robototechnocs.

RESUME

1. New interfaces on a module boards (PCIe, VME) are shifted to new high-speed serial interconnections for I/O peripherals subsystems.
2. System interconnect (PCIe, SCI, Infiniband) are based on multi-gigabit switches system architectures for new multiprocessor and cluster design for high-performance computing system, SAN, HPC and Data Centers.
3. Very compact modular system can be constructed on the base on micro TCA and AMC modules, inserted in special compact shelf, which can be more effective for many applications..

REFERENCES

[1] V,I.Vinogradov. Advanced diversified compact modular Architectures for convergenced high-performance computer, network and telecommunication systems.Preprint INR–1171/ 2006. Moscow

[2] V.I. Vinogradov. *Advanced High-performance computer systems architecture. EuroMedIm-2006. CERN Conference. Marseille. France

..

The ACOP Family of Beans

Philip Duval and Honggong Wu, DESY MST, Hamburg, Germany
Igor Kriznar, COSYLAB

Abstract

The current ACOP (Advanced Component Oriented Programming) [1] controls set consists of an ActiveX chart control and the equivalent chart bean [2]. ACOP has enjoyed great success as a rapid application design tool for rich clients in control system applications. It is a narrow-interface control which offers design-time browsing of the control system to expedite data acquisition and numerous data rendition features to satisfy needs of the most demanding control application developers, to the extent that a two-dimensional chart is what is desired.

We now extend the functionality of ACOP Java bean in several key ways. We incorporate automatic 'office-like' drag-and-drop (DnD) of the meta-information behind any displayed data. We allow design-time configuration of "simple" clients (as opposed to "rich" clients), where a finished application can be configured without writing a single line of code, if that is what is desired. We extend the ACOP family of beans to include an ACOP compatible Label, Slider, Table, and Image to go along with the ACOP Chart.

1 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.

2 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 devices 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 will allow the setting of all such transport parameters.

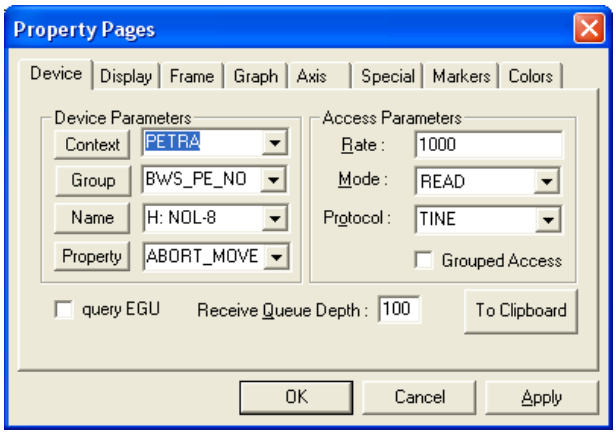


Fig 1. Customizer for ACOP transport bean

If more than one displayer within the same application are connected to the same endpoint (for instance, a chart and label are both connected to the

beam current), this information is passed seamlessly from one ACOP displayer to the other.

3 ACOP GRAPHIC BEANS

The ACOP family of graphical displayers consists of a two dimensional chart, label, slider, table, and image container. They are extended from their counter parts in the Swing graphical widget set. In each of these displayers, the ACOP transport bean is referenced so that control system browsing of available endpoints is automatically supported.

3.1 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. If an ACOP application user makes run-time configuration changes, he has the option of making these changes persistent.

At design time, 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 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 transport endpoints directly to the displayer without writing a single line of code.

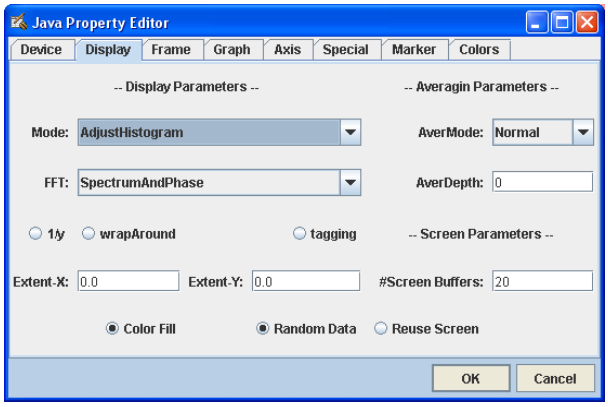


Fig 2. Customizer for ACOP chart bean.

3.2 Meta-Data Popup

ACOP graphic beans all respond automatically to a '<SHIFT>-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.

3.3 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, using the original display colors and scale settings. Dropping into another ACOP chart in design time will apply the dropped settings. To this end, an ACOP displayer in design time will accept endpoint information dragged for instance from Notepad, if it can be interpreted. Dragging from one ACOP display to another 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 color, but not any additional scale settings. The above case of an ACOP connected to the beam current and beam lifetime will pass the scale settings to the ACOP label, but the label will not be able to use these settings and therefore jettison them.

In addition, if for instance a multi-channel array is being displayed in an ACOP chart as a histogram, 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.

CONCLUSION

We have made much progress in developing the ACOP family of beans, but much work remains to be done. Most of the emphasis has been on the ACOP Chart bean, since it is the most complex. When finished the addition of any new displayer to the ACOP

family of beans should be no more complicated than extending the displayer to include the ACOP DnD, meta-displayer, and customizer classes and to reference the ACOP transport bean. Java applications using the ACOP family of beans should be able to pass information to one another in a framework independent way. To this end, care should also be taken to insure that future ACOP toolkits (for instance ACOP .NET controls) can also seamlessly pass and interpret DnD information among themselves and their brethren.

The complete ACOP graphic toolkit should be ready for the initial phase of client development for PETRA III in the middle of 2007.

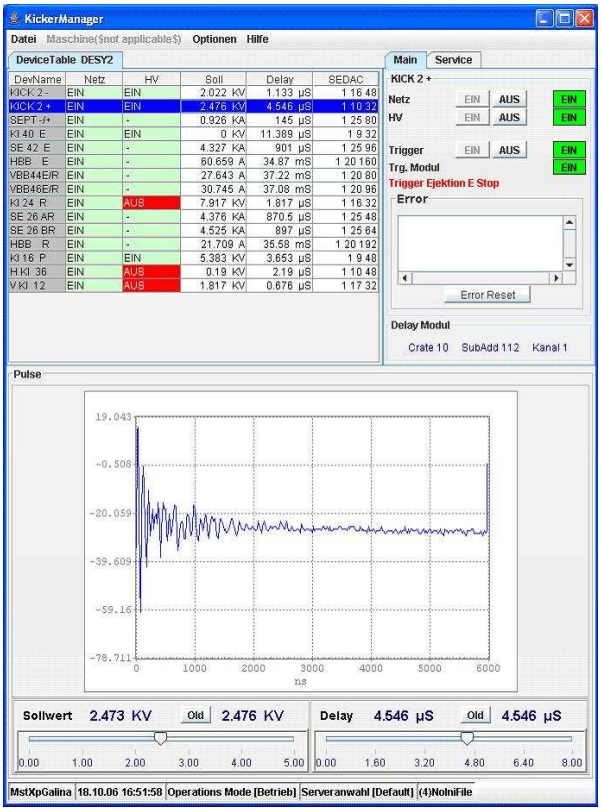


Fig 3. An ACOP application showing the Kicker Settings for the DESY2 accelerator, using the ACOP chart and table displayers.

REFERENCES

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

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

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

ACS – AN OPEN SOURCE CONTROL SYSTEM INFRASTRUCTURE

K. Žagar[#], G. Chiozzi[*], M. Šekoranja[#], H. Sommer[*], M. Pleško[#], B. Jeram[*], A. Caproni[*], R. Cirami[§], P. Di Marcantonio[§]

Abstract

Today, ACS is a mature control system infrastructure. Since its inception in year 2000, more than 30 person-years of effort have been put into its development, maintenance and continuous improvement. Though primarily intended for the *Atacama Large Millimeter Array* (ALMA) radio-telescope (whence the abbreviation ACS – *ALMA Common Software*), it has been found useful in other projects as well, ranging from distributed control of a synchrotron light source to an application server for business applications.

In this paper, a technical overview of ACS is given and its fundamental concepts are explained. Then, lessons learned in the last half-a-decade are presented, both from technological as well as organizational perspective.

INTRODUCTION

The *Atacama Large Millimeter Array* (ALMA) radio interferometer is currently being built in the Chajnantor area of the Atacama Desert in Chile. Renown for its extremely dry weather conditions, the site is almost ideal for astronomical observations. The facility will consist of multiple radio antennas that will be collecting large quantities of scientific data, which will then be processed in near real-time through configurable processing pipeline and with scientific results being archived. The ALMA facility consists of dozens of instruments. All of these subsystems need to be controlled…

To consolidate the effort required to build a control system for such a complex facility, development of *ALMA Common Software* (ACS) started in year 2000. The goal of ACS was to give all control system developers a common platform to work on, so that not everyone would have to re-invent mechanisms for remote procedure calls, configuration, alarms, logging, etc.

Since, ACS has found its application not only in ALMA, but also other astronomy instruments currently being built. Furthermore, ACS has been found useful for control of synchrotrons and deployment of business applications.

As ACS has been under development for more than 6 years, the core concepts are now implemented in a very stable fashion. The total effort allocated to ACS development by the ALMA project is on the order of 30 man years, but there have been contributions to ACS from its other users as well.

Cosylab, Ljubljana, Slovenia
* European Southern Observatory, Garching, Germany
§ INAF Osservatorio Astronomico di Trieste, Italy

ACS CONCEPTS

ACS attempts to provide solutions to most issues encountered in distributed control system. This section explains some of them.

Manageable Distributed Component Model

ACS is built around a model where functionality (e.g., control of a particular device through its specific protocol) is encapsulated in *components*. A component exposes a functional interface which models the actions that the device can perform and the control/monitor points (properties) associated with the device. The approach is thus highly-modular, with components being the building blocks.

Components execute within containers (see Figure 1). Containers are processes that are deployed at hosts across a computer network. The hosts are expected to be physically connected to the devices under control of components, placed within the host's container.

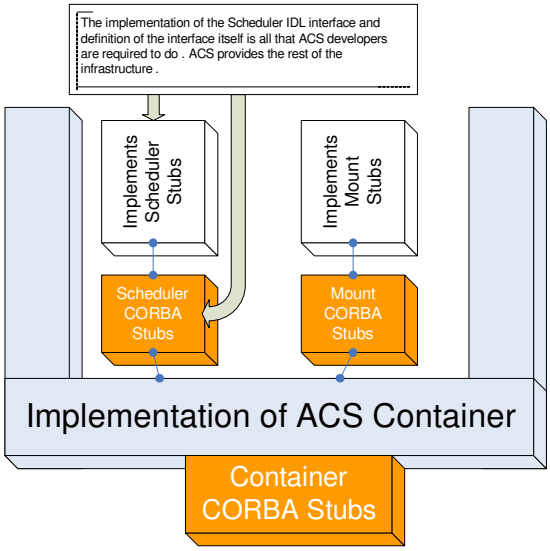


Figure 1: Architecture of the component/container model.

A component is accessible over the network as a CORBA object. This makes it possible to manipulate the device remotely, and serves as the basis of the distributed control system.

Component implements functionality such as access to a particular device, state machine for control, etc. On the other hand, the component is completely unaware of how it is accessed – it can be local access from a unit test, or a remote access via CORBA. Also, the component may use facilities such as logging, raising of alarms, etc.

A component can be placed in any container. There is a central manager, which keeps track of placement of components. Also, the manager is a kind of a naming service, whom clients consult to obtain references to components.

The container hides all the details of CORBA and ACS (logging, remote procedure calls, event distribution, …). The component developer can thus focus on the functionality he/she is to provide.

XML Configuration Database

All configuration data is stored in an XML configuration database. Every type of a component defines a structure of the XML data that is needed to configure it. The structure is defined with the standard XML *Schema Definition Language* (XSD), and is used for checking the correctness of the configuration database. Also, a configuration database tool exists which uses the XSD of a particular record to deduce the data fields that a particular component relies upon. Also, XSD may be used to convey default values of fields.

The configuration data is designed to be more or less static (that is, the configuration database does not contain run-time information, such as values of control points). Nonetheless, it is possible to change the configuration database in runtime, and force the affected components to reload for changes to take effect.

Configuration database is available to the rest of the system as a CORBA object implementing a well-defined IDL. The manager and the containers make use of this interface to access the configuration data, and pass it to components.

Physical storage of the configuration data is up to the implementation of the configuration database. By default, XMLs are stored in a hierarchical database on a file system. Alternative implementations also exist, e.g., for storing configuration data in a relational database, where indexing can be used to improve retrieval performance.

CORBA-Based RPC

CORBA is used as the middleware to facilitate remote procedure calls. CORBA is a well defined standard with many implementations (ORBacus, JacORB, TAO) and services (naming service, interface repository, notification service, etc.).

CORBA is designed to facilitate cross-platform and cross-language communication. Thus, it allows ACS to run on Linux and Windows platforms, as well as some real-time operating systems. As programming languages, ACS components and clients can be written in Java, C++ or Python.

Alarm System

Initial versions of the ACS implemented the alarm system in a point-to-point manner, where a party interested in alarms (e.g., a GUI console) would register callbacks with all the entities capable of producing alarms. Once an alarm would occur, the interested party would be notified through a callback.

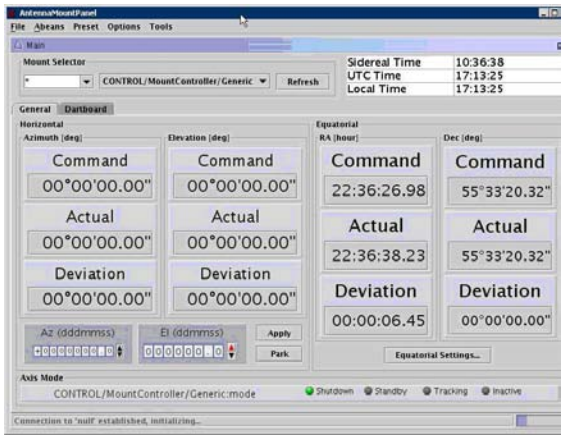


Figure 2: User interface for controlling the antenna mount (Java and Abeans).

This approach was found not to scale well as the amount of network connections was excessive for real-life systems – an issue of particular importance during alarm avalanches, where one fault state is a cause to several others. Also, the alarm system was difficult to configure.

Therefore, the ACS team decided to integrate the alarm system being built at CERN for the *Large Hadron Collider* (LHC) called LASER [3]. LASER uses publisher-subscriber paradigm for alarm distribution, implemented using JMS interface to *CORBA Notification Service*: the alarm service subscribes to alarm event channels, whereas alarm sources publish into those channels. When the alarm service receives an alarm, it processes it, e.g., to detect alarm avalanches and reduce the number of alarms actually passed on to the operators.

Bulk Data Transfer

CORBA RPC is not the best choice for transmission of large amounts of data because of the marshalling overhead associated with CORBA calls. However, CORBA offers an audio/video streaming service, where CORBA is used to define the manner of transport, and the bulk transport is carried out directly via TCP/IP or UDP/IP.

In ACS, the CORBA streaming service is leveraged to build a *Bulk Data Transfer* service [4]. Apart from the point-to-point transfer, this service features a *Distributer* component to which bulk data sources stream data. The *Distributer* then streams the data to final sinks of data. This decoupling prevents the sources from being overloaded if too many sinks are connected.

Event Distribution and Distributed Logging

ACS makes use of CORBA's *Notification Service* for delivery of events that are of interest to a wider audience (e.g., initiation of a phase in the lifecycle of astronomical observation).

The *Notification Service* is also used to implement distributed logging. A well-designed distributed logging infrastructure is mandatory in distributed control system in order to facilitate diagnostics or validate correctness of

operation. A well-defined logging API is provided to the application developers, which ensures delivery of log entries to a central logging service. Log entries are cached locally and delivered in batches in order to improve performance. The logging service uses a permanent storage to record the log entries. GUI for viewing and filtering the log entries is available – this GUI connects to the central log service.

Applications

For composing graphical user interfaces, developers can choose the framework they feel most comfortable with – Abeans/CosyBeans (see Figure 2) or Swing in Java, Qt in C++, etc. This allows the developers to use a most appropriate tool for the task at hand.

ACS USER COMMUNITY

In the last few years, ACS has been used extensively in the field: the ALMA prototype antennas have been running a first version of the ALMA Control Software and are now running the latest ALMA software baseline release and the end-to-end ALMA software is integrated and tested as a whole on a periodic basis, with two official releases per year,. This provides good testing and feedback concerning the ACS global infrastructure, performance and tools from the operational and deployment points of view. ACS-based software is also used in various laboratories by the teams developing hardware and devices for ALMA.

ACS is used at facilities other than ALMA as well. For example, *Max-Planck-Institut für Radioastronomie* is using ACS for controlling the 12 meter radio-telescope *Atacama Pathfinder Experiment* (APEX), which is now already in operation. Similarly, synchrotron *Angströmquelle Karlsruhe* (ANKA) is also using ACS in production, and is regularly updating to the latest stable release.

Nearing operation are the following facilities whose control system is ACS based:

- 1.5m Hexapod Telescope (HPT) of the *Ruhr-University Bochum*.
- The 40m telescope of the *Observatorio Astronómico Nacional*.
- The 32m radio *Sardinia Radio Telescope*.

Commercial Support and Applications

Cosylab is using ACS in distributed, high-performance production, high-availability environments of business applications (e.g., image servers for Geographical Information Systems, GIS).

Apart from using the ACS infrastructure, Cosylab is also actively participating in its ongoing development, and is capable of offering support, consulting and training to other existing or prospective users of ACS.

Cosylab's embedded I/O controller, *microIOC* (http://www.microioc.com) comes with ACS pre-installed. This product enables control system integrators to quickly set-up nodes of a control system, without having to configure the operating system or ACS.

CONCLUSION AND LESSONS LEARNED

In the last 6 years, several lessons have been learned developing, using, maintaining and deploying the ACS.

ACS has succeeded in becoming a one-stop-shop for all developers' middleware needs: from remote procedure calls, through configuration, logging, centralized management, error handling, and to building user interfaces.

The list of hardware devices that can be integrated into ACS is growing. The support is not only being added for custom, ALMA-specific hardware (antenna mounts, interferometers, receivers, etc.), but also for widely available products such as joystics and cameras.

Standards-based approach has shown to be beneficial several times already. As some implementations of CORBA services were found not to meet all requirements (e.g., in terms of performance or support of the standard), they were exchanged with competing implementations, without much an adverse effect to the rest of the system.

In ACS, all components are available through wide interfaces which clearly expose the functionality of the component. This way, compile-time checking of interface-level compatibility is possible, allowing for early detection of defects.

We have found a rigorous development process (nightly builds, extensive unit tests, etc.) essential to manage development of a complex system such as ACS.

REFERENCES

[1] K. Žagar et al. "ACS – Overview of Technical Features", ICALEPCS'03, Gyeongju, South Korea, October 2003.

[2] G. Chiozzi et al. "Application development using the ALMA common software", Proc. SPIE, Vol.6274, Astronomical Telescopes and Instrumentation, Advanced Software and Control for Astronomy, Orlando, USA, May 2006.

[3] A. Caproni, K. Sigerud, K. Zagar. "Integrating the CERN Laser Alarm System with the Alma Common Software", Proc. SPIE, Vol.6274, Astronomical Telescopes and Instrumentation, Advanced Software and Control for Astronomy, Orlando, USA, May 2006.

[4] R. Cirami, P. Di Marcantonio. "Bulk Data Transfer Distributer: a high performance multicast model in ALMA ACS", Proc. SPIE, Vol.6274, Astronomical Telescopes and Instrumentation, Advanced Software and Control for Astronomy, Orlando, USA, May 2006.

LIST OF PARTICIPANTS

Anicic, Damir Paul Scherrer Institut, Villigen PSI, 5232 Switzerland
damir.anicic@psi.ch

Bacher, Reinhard DESY, Notkestr. 85, Hamburg, 22603 Germany
reinhard.bacher@desy.de

Baer, Ralph GSI Darmstadt, Planckstr. 1, Darmstadt, 63069 Germany
R.Baer@gsi.de

Bevins, Brian Jefferson Lab, 12000 Jefferson Ave, Newport News,
VA 23606, USA
bevins@jlab.org

Bickley, Matthew Jefferson Lab, 12000 Jefferson Ave, Newport News,
VA 23606, USA
bickley@jlab.org

Bieler, Michael DESY, Notkestr. 85, Hamburg, 22603 Germany
bieler@desy.de

Bodenstein, Ryan UVA, 12050 Jefferson Ave, Newport News,
VA 23606, USA
cyanb@jlab.org

Bolkhovityanov, Dmitry BINP, Lavrentyeva 11, Novosibirsk, 630090 Russia
D.Yu.Bolkhovityanov@inp.nsk.su

Carlino, Isadoro Jefferson Lab, 12000 Jefferson Ave, Newport News,
VA 23606, USA
carlino@jlab.org

Catani, Luciano INFN-Roma Tor Vergata, via della Ricerca Scientifica 1,
Roma, 00133 Italy
luciano.catani@roma2.infn.it

Chaize, Jean-Michel ESRF, 6 rue J Horowitz, Grenoble , 38000 France
chaize@esrf.fr

Chevtsov, Pavel Jefferson Lab, 12000 Jefferson Ave, Newport News,
VA 23606, USA
chevtsov@jlab.org

Chevtsov, Sergei SLAC, Stanford, CA, USA
chevtsov@slac.stanford.edu

Clausen, Matthias DESY, Notkestr. 85, Hamburg, 22607 Germany
Matthias.Clausen@desy.de

Curry, Doug Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
dcurry@jlab.org

Di Pirro, Giampiero Laboratori Nazionali di Frascati – INFN, Via Enrico Fermi 40, Frascati (RM), 00044 Italy
giampiero.dipirro@lnf.infn.it

Dickson, Richard Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
dickson@jlab.org

Duval, Philip DESY, Notkestr. 85, Hamburg, 22607 Germany
Philip.Duval@desy.de

Ehrlichmann, Heiko DESY, Notkestr. 85, Hamburg, 22607 Germany
Heiko.Ehrlichmann@desy.de

Epps, Michael Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
epps@jlab.org

Evans, Richard Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
revans@jlab.org

Fujita, Jiro Creighton University, 2500 California Plz., Omaha, NE 68178, USA
jiro@creighton.edu

Furukawa, Yukito SPring-8/JASRI, 1-1-1 Koto, Sayo, Hyogo 679-5198 Japan
furukawa@spring8.or.jp

Gaio, Giulio Elettra, S.S. 14 km 163.5 in AREA Science Park, Basovizza, Trieste, 34012 Italy
giulio.gaio@elettra.trieste.it

Gajsek, Rok Cosyab, Slovenia
rok.gajsek@cosylab.com

Giacchini, Mauro	INFN, Legnaro, 35020 Italy mauro.giacchini@lnl.infn.it
Herb, Steve	DESY, Notkestr. 85, Hamburg, 22607 Germany Steve.Herb@desy.de
Hutton, Andrew	Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA andrew@jlab.org
Höppner, Klaus	GSI Darmstadt, Planckstr. 1, Darmstadt, 64291 Germany k.hoeppner@gsi.de
Jordan, Kevin	Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA jordan@jlab.org
Joyce, Michele	Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA erb@jlab.org
Kamikubota, Norihiko	KEK, 1-1 Oho, Tsukuba, Ibaraki 305-0801 Japan norihiko.kamikubota@kek.jp
Kammering, Raimund	DESY, Notkestr. 85, Hamburg, 22607 Germany raimund.kammering@desy.de
Karnaev, Sergey	BINP, Lavrentyeva 11, Novosibirsk, 630090 Russia karnaev@inp.nsk.su
Katoh, Tadahiko	KEK, 1-1 Oho, Tsukuba, Ibaraki 305-0801 Japan tadahiko.katoh@kek.jp
Keesee, Marie	Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA keesee@jlab.org
Kerscher, Harald	MED PT PLM B
Kleines, Harald	Forschungszentrum Juelich, Leo-Brand-Str., Juelich, 52425 Germany h.kleines@fz-juelich.de
Kosuge, Takashi	KEK, 1-1 Oho, Tsukuba, Ibaraki 305-0801 Japan takashi.kosuge@kek.jp

Labudda, Andreas DESY, Notkestr. 85, Hamburg, 22607 Germany
andreas.labudda@desy.de

Lahti, George Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
lahti@jlab.org

Larrieu, Theo Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
theo@jlab.org

Lawson, Greg SNS/ORNL, USA
lawsongs@ornl.gov

Lomperski, Mark DESY, Notkestr. 85, Hamburg, 22607 Germany
Mark.Lomperski@desy.de

McGuckin, Theo Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
tsm@jlab.org

Mezger, Anton Paul Scherrer Institut, Villigen PSI, 5232 Switzerland
anton.mezger@psi.ch

Moore, Wesley Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
wmoore@jlab.org

Morozov, Ivan BINP, Lavrentyeva 11, Novosibirsk, 630090 Russia
morozov@inp.nsk.su

Nakatani, Takeshi JAEA, Japan
takeshi.nakatani@j-parc.jp

Nishimura, Hiroshi LBNL, MS 80-101, Univ. of California, Berkeley, CA 94720 USA
H_Nishimura@lbl.gov

Ohata, Toru SPring-8/JASRI, 1-1-1 Koto, Sayo,
Hyogo 679-5198 Japan
ohata@spring8.or.jp

Okay, Noel Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
okay@jlab.org

Osanai, Akihiro Kyoto University, Japan
osanai@post3.rri.kyoto-u.ac.jp

Pace, Elisabetta Laboratori Nazionali di Frascati – INFN, Via Enrico Fermi 40, Frascati (RM), 00044 Italy
elisabetta.pace@lnf.infn.it

Pal, Sarbajit Variable Energy Cyclotron Centre, India
sarbajit@veccal.ernet.in

Payne, Chris TRIUMF, Canada
chris.payne@triumf.ca

Penno, Marek DESY, Platanen Allee 6, Zeuthen, 15738 Germany
marek.penno@desy.de

Plesko, Mark Cosylab, Slovenia
mark.plesko@cosylab.com

Podborsek, Aljaz Cosylab, Slovenia
aljaz.podborsek@cosylab.com

Purcell, David SNS, USA
purcelljd@sns.gov

Quock, Deborah ANL, 9700 South Cass Avenue, Argonne, IL 60439, USA
quock@aps.anl.gov

Ristau, Uwe EMBL-Hamburg, Notkestrasse 83, Hamburg, 22603 Germany
ristau@embl-hamburg.de

Sarrazin, Diane Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
sarrazin@jlab.org

Sexton, Dan Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
dsexton@jlab.org

Tanigaki, Minoru Research Reactor Institute, Kyoto University, 2-1010 Asashiro-nishi, Kumatori, Osaka 590-0494 Japan
tanigaki@rri.kyoto-u.ac.jp

Tengsirivattana, Chaivat Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
chaivat@jlab.org

Vasiliev, Dmitri IHEP

Vinogradov, Vyacheslav INR RAS, Prosp 60-Anniv. of October 7 a, Moscow, 117312 Russia
vin@inr.ru

Watson, Chip Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
watson@jlab.org

White, Karen Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
karen@jlab.org

Wolin, Elliot Jefferson Lab, USA
wolin@jlab.org

Wu, Honggong DESY, Notkestr. 85, Hamburg, 22607 Germany
hong.gong.wu@desy.de

Yamashita, Akihiro SPring-8, 1-1-1 Koto, Sayo, Hyogo 679-5198 Japan
aki@spring8.or.jp

Yan, Jianxun Jefferson Lab, 12000 Jefferson Ave, Newport News, VA 23606, USA
yanjx@jlab.org

Zambon, Lucio Elettra, Basovizza, Trieste, 34012 Italy
lucio.zambon@elettra.trieste.it

AUTHOR INDEX

A

Abe, N., 26
Anderson, J., 93
Anicic, D., 61
Arnold, N., 93

B

Bacher, R., 7
Bartkewicz, P., 108
Bassato, G., 116
Bellaveglia, M., 66
Bhattacharjee, T., 40
Bhole, R.B., 40
Bickley, M., 90, 99
Bieler, M., 82
Blokland, W., 136
Bobnar, J., 120
Bolkhovityanov, D., 111
Brinkmann, A., 82
Brnicky, M., 15
Burns, J., 15

C

Caproni, A., 152
Carlino, I., 86
Catani, L., 66, 78
Chaddha, N., 40
Chaize, J.-M., 3
Cherney, M., 15
Chevtsov, P., 30, 90
Chevtsov, S., 69
Chiozzi, G., 152
Cianchi, A., 66
Cirami, R., 152
Clausen, M., 102
Clemons, D., 93

D

Dasgupta, S., 40
Detert, S., 53
Di Marcantonio, P., 152
Di Pirro, G., 66
Dohan, D., 93
Dong, H., 129
Drochner, M., 53
Duval, P., 17, 20, 72, 114, 149

F

Filippetto, D., 66
Forchi, V., 23
Froehlich, G., 120
Fujita, J., 15
Fukui, T., 11
Furukawa, Y., 105

G

Gaio, G., 23
Gajsek, R., 43
Giacchini, M., 116
Golob, D., 49
Gorbunov, Y., 15
Grevsmuehl, T., 39
Grippo, A., 46

H

Habjan, I., 117
Hasanovic, A., 33, 49
Hatje, J., 102
Herb, S., 72, 108
Hirono, T., 11
Hoeppner, K., 120
Hosoda, N., 11

I

Ikeda, M., 26
Ishizawa, Y., 96

J

Jansa, G., 43
Jeram, B., 152
Jordan, K., 46

K

Kamenik, J., 58
Kaplin, V., 123
Karnaev, S., 123
Kijima, Y., 26
Kitamura, M., 11
Kobal, M., 43
Koehler, W., 39
Kolaric, P., 58
Korhonen, T., 61
Kosuge, T., 20
Kleines, H., 53
Krause, U., 120
Kretzschmann, A., 39
Kriznar, I., 49, 120, 149

L

Labudda, A., 126
Lahti, G., 129
Leclercq, N., 23
Leich, H., 39
Liyu, A., 136
Lomperski, M., 84
Lonza, M., 75

M

Maesaka, H., 11
Masuda, T., 11
Matsushita, T., 11
McGuckin, T., 131
Meshkov, O., 123
Mezger, A., 61
Miginskaya, E., 134
Mishima, K., 26
Moeller, M., 102
Mori, Y., 26
Moore, W., 46
Morozov, I., 123, 134

N

Nagatani, Y., 20
Nigorikawa, K., 20
Nishimura, H., 37, 56

O

Ohata, T., 11, 96, 105
Ohshima, T., 11
Osanai, A., 139
Otake, Y., 11

P

Pace, E., 66
Pal, S., 40
Patton, J., 136
Payne, C., 63
Pelaia, T., 136
Penno, M., 39
Petrosyan, B., 39
Plesko, M., 33, 49, 58, 120, 152
Plotnikova, O., 123
Podborsek, A., 49
Purcell, J.D., 136

Q

Quock, D., 93

R

Roy, A., 40

S

Sabjan, R., 49
Scafuri, C., 23
Schaa, V.R.W., 120
Seeberger, T., 129
Sekoranja, M., 117, 152
Sexton, D., 46
Shiroya, S., 26
Smaluk, V., 123
Sommer, H., 152
Suxdorf, F., 53

T

Takamiya, K., 26
Takeshita, T., 26
Takeuchi, M., 11, 96
Tanaka, R., 11
Tanigaki, M., 26
Thomen, R., 15
Timossi, C., 37, 56
Trowitzsch, G., 39
Tsukanov, V., 134

V

Vermeulen, D., 61
Verstovsek, I., 58
Vinogradov, V., 142, 145
Volkov, A., 134

W

Waggoner, W., 15
Wenndorff, R., 39
Wilgen, J., 114
Wu, H., 17, 149

Y

Yamashita, A., 11, 96
Yan, J., 46
Yoshino, H., 26

Z

Zagar, K., 33, 117, 152
Zambon, L., 75
Zhukov, A., 136
Zhuravlev, A., 123
Zobjack, U., 82